Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Н. Н. Пустовалова, Н. В. Пацей

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ
В 2-х частях
Часть 2

Лабораторный практикум
для студентов IT-специальностей

Минск 2018

ПРЕДИСЛОВИЕ

Практикум содержит задания для выполнения лабораторных работ на основе приложения **Microsoft Visual Studio**. В каждой работе имеются краткие теоретические сведения по рассматриваемым вопросам.

Преподаватель определяет, какие лабораторные работы должны выполнять студенты и в каком объеме. Предполагается, что выполнение большинства лабораторных работ занимает у студентов два академических часа.

Задания для выполнения лабораторных работ содержат кнопки, при нажатии на которые открываются тесты, предназначенные для контроля знаний студентов. Тестирование происходит по команде преподавателя и занимает несколько минут.

Для работы тестирующих программ предварительно в приложении Word надо разрешить использование макросов. При этом тексты ответов на формах располагаются каждый раз случайным образом, и ответить на вопросы можно только один раз, так как после нажатия на кнопку «Результаты» форма с вопросами и вариантами ответов исчезает.

При **оформлении отчетов по лабораторным работам** необходимо использовать приложение **Word**. Каждый отчет должен содержать название работы, условия задач, блок-схемы алгоритмов, листинги разработанных программ, результаты выполнения. В верхнем колонтитуле записывается фамилия студента и номер группы, в нижнем – номера страниц. Шрифт – 10 или 12, интервал – одинарный, поля – по 1,5 см. Все отчеты сохраняются в одном файле.

ОГЛАВЛЕНИЕ

Лабораторная работа № 1. Указатели на функции

Задание	Краткие теоретические сведения
1. Изучить использование *указателя на функцию*, выполнив программу, записанную в правой части.	**Указателем на функцию** называется **переменная**, которая содержит **адрес функции** (адрес точки входа в функцию).

```
#include <iostream>
using namespace std;
int f1 (char *s)
{  cout << s <<" \nВыполняется тест №1" << endl;
   return 1;
}
int f2 (char *s)
{  cout << s<<" \nВыполняется тест №2" << endl;
   return 2;
}
void callfun (char *s, int (*f)(char *))
{  cout << " f() = " << f(s) << '\n';
}

int main ()
{  setlocale(0, "Rus");
   callfun("Тест №1 вызван", f1);
   callfun("Тест №2 вызван", f2);
}
```

Пример. Пусть для некоторого проекта надо выполнить последовательность тестов, реализуемых функциями **f1** и **f2**.

В представленной программе функция **callfun** предназначена для вызова той тестирующей программы, имя которой передается из главной функции в указатель на функцию:

int (*f)(char *).

2. В программе, приведенной в правой части, демонстрируется использование *указателя на функцию* для вычисления площади криволинейной трапеции.

```
#include <iostream>
float integ(float(*) (float), float, float, float);
//прототип
float f(float);      //прототип

int main()
{     float z;
      z = metodPrjam(f, (float)0.0, (float)10.0,
(float)0.01);
      std::cout << "Result=" << z;
}

float metodPrjam(float(*fun) (float), float a,
float b, float h)
{     float x, s = 0.0;
      x = a + h;
      while (x <= b)
      {     s += h*fun(x);
            x = x + h;
      }
      return s / 2;
}

float f(float x)
{     return (2 * x + 5);
}
```

Площадь криволинейной трапеции ограничена графиком функции **f(x) = 2 * x + 5**, прямыми **x = 0**, **x = 10** и осью **OX**.

В данной программе из главной функции происходит обращение к функции **metodPrjam**, реализующей для аычисления площади метод правых прямоугольников.

Среди формальных параметров функции **metodPrjam** записан указатель на функцию **float(*f) (float)**, в который передается адрес функции **f**, содержащей операторы вычисления функции **f(x) = 2 * x + 5**.

3. Выполнить программу, содержащую функцию с *переменным числом параметров*.

Записать условие задачи.

При вызове функции с *переменным числом параметров* задается любое требуемое число аргументов.

```
#include <iostream>
int sum(int n, ...);
using namespace std;

void main()
{
      cout << sum(6, 4, 5, 1, 2, 3, 0)<<std::endl;
      cout << sum(2, 34, 4445);
}

int sum(int n, ...)
{
      int *p = &n;
      int sum = 0;
      for (int i = 1;  i <= n;  i++)
            sum += *(++p);
      return sum;
}
```

В объявлении и определении такой функции переменное число аргументов задается многоточием в конце списка формальных параметров или списка типов аргументов. При этом должен быть указан как минимум один обычный параметр.

4. В соответствии со своим вариантом *отделить корни* двух уравнений и вычислить их методом *дихотомии* для исходных данных из таблицы, приведенной ниже. Точность вычислений принять равной **e** = 0,001 для всех вариантов.

Операторы метода вычисления корня оформить в виде функции пользователя, уравнения записать также в виде функций пользователя. В основной программе предусмотреть ввод исходных данных, обращение к функции, реализующей метод дихотомии. В процессе выполнения программы определить корни двух уравнений. Использовать *указатель на функцию*.

№ варианта	Функции f(x)	№ варианта	Функция f(x)	№ варианта	Функция f(x)
1	$x^2 + 4x - 2$, $\sin(x) + 0{,}1$	7	$x^3 + 2x - 4$, $x^2 - 1$	13	$5x - 1 + x^3$, $x^2 + 1 / x$
2	$x^3 + x - 3$, $\cos^3(x)$	8	$x^3 + 3x - 1$, $e^x - 4$	14	$e^x + x - 4$, $x^2 - 4$
3	$\cos(x) + x - 7$, $e^x - 1 / x$	9	$x^3 + x - 4$, $1 - x^2$	15	$x^3 + x - 2$, $\sin(x) + 0{,}3$
4	$x^3 + 2x - 1$, $e^x - 2$	10	$\sin(x) + x^3$, $0{,}4 + x^3$	16	$\sin(x) + 2 + x$, $2 + x^3$
5	$e^x - 3 - 1 / x$, $0{,}2 - x^2$	11	$e^x + 2x^2 - 3$, $x^3 + 3$		
6	$2 - x^2 + x$, $\sin^2(x) + 0{,}2$	12	$2x + x^3 - 7$, $e^x + 2x$		

5. В соответствии со своим вариантом написать программы по условиям задач из таблицы ниже. Программа должна содержать функцию пользователя с *переменным числом параметров* и не менее трех обращений к ней с различным количеством параметров.

№ варианта	Условие задачи
1	Написать функцию **fmin** с переменным числом параметров, в которой определяется минимальное из чисел типа **int**.
2	Написать функцию **fsum** с переменным числом параметров, в которой определяется сумма чисел типа **int** по формуле: ***S=a1*a2+a2*a3+a3*a4+ . . . .***

№ варианта	Условие задачи
3	Написать функцию **fmax** с переменным числом параметров, которая находит минимальное из чисел типа **int** или из чисел типа **double**, тип параметров определяется с помощью первого параметра функции.
4	Написать функцию **days** с переменным числом параметров, которая находит количество дней, прошедших между двумя датами (параметрами функции являются даты в формате «дд.мм.гг».
5	Написать функцию **kvadr** с переменным числом параметров, которая определяет количество чисел, являющихся точными квадратами (2, 4, 9, 16,…) типа **int**.
6	Написать функцию **as** с переменным числом параметров, которая находит сумму чисел типа **int** по формуле: ***S=a1*a2-a2*a3+a3*a4-. . . .*** .
7	Написать функцию **mn** с переменным числом параметров, которая находит минимальное из чисел типа **int** или из чисел типа **double**, тип параметров определяется с помощью первого параметра функции.
8	Написать функцию **prost** с переменным числом параметров, которая находит все простые числа из нескольких интервалов. Интервалы задаются границами **a** и **b**. С помощью этой функции проверить несколько интервалов.
9	Написать функцию, которая в предложении подсчитывает количество символов в слове максимальной длины (слова разделяются пробелами). Опробовать работу функции на нескольких предложениях.
10	Написать функцию, которая находит в строке самое первое (по алфавиту) слово. С ее помощью реализовать размещение слов в выходной строке в алфавитном порядке.
11	Написать функцию с переменным числом параметров, которая находит минимальное значение матрицы. С ее помощью найти минимальные значения в n матрицах.

№ варианта	Условие задачи
12	Написать функцию, проверяющую есть ли отрицательные элементы в заданном одномерном массиве размерностью **n**. Удалить из массива все отрицательные элементы, удаленный элемент заполняется нулем и переносится в конец массива.
13	Написать функцию для вычисления суммы элементов квадратной матрицы, которые расположены ниже главной диагонали. С ее помощью найти максимальное значение такой суммы в n матрицах.
14	Написать функцию **compr**, которая «сжимает» строку, удаляя из нее все пробелы. С ее помощью сжать различные строки.
15	Написать функцию с переменным числом параметров для перевода чисел из десятичной системы счисления в двоичную. С помощью этой функции перевести различные числа из десятичной системы счисления в двоичную.
16	Написать функцию, которая определяет, есть ли в тексте слова, начинающиеся с буквы **a**, и сколько таких слов. С помощью этой функции проверить несколько строк.

Тест "Массивы и ссылки при работе с функциями"

В начало практикума

Лабораторная работа № 2. Работа с файлами на языке С

Файл – это набор данных, размещенный на внешнем носителе и рассматриваемый в процессе обработки как единое целое. Каждый файл завершается маркером конца файла (end-of-file marker или EOF) или указанным числом байтов, записанным в служебную структуру данных.

В программу должна быть включена директива **#include <stdio.h>**.

Задание	Краткие теоретические сведения
1. Изучить *форматированный* ввод и вывод при работе с *текстовыми* файлами на языке С, выполнив программу, записанную в данном пункте. Дополнить программу операторами чтения файла и вывода на экран его содержимого.	Файл открывается с помощью функци **fopen**, общий вид которой: **fopen(const char *filename, const char *mode)** Здесь **filename** – имя файла, **mode** – режим открытия файла. Режимы открытия файла: **r** – открывает файл для чтения (если файл не существует, то вызов завершается ошибкой); **w** – открывает пустой файл для записи (если файл существует, его содержимое удаляется); **a** – открывает файл для записи в конец файла (создает файл, если он не существует); **r+** – открывает для чтения и записи в существующий файл; **w+** – открывает пустой файл для чтения и записи (если файл существует, его содержимое удаляется); **a** – открывает файл для чтения и добавления в конец. Могут быть также добавлены символы **t** (открыть файл в текстовом режиме) либо **b** (открыть файл в бинарном режиме). Используется также функция, общий вид которой: **fopen_s(FILE** pFile, const char *filename, const char *mode)**, где **pFile** – указатель на файловый указатель. Файл закрывается с помощью функции **fclose**.

```c
#include <stdio.h>
#include <iostream>
int main()
{   setlocale(LC_ALL,"");
    int a; errno_t err;
    FILE *f;
    err = fopen_s(&f,"a.txt", "w");
    if (err != 0)
    {    perror("Невозможно создать файл\n");
         return EXIT_FAILURE;
    }
    for(a = 0;  a < 70;  a += 3)
    {   fprintf(f, "%d, ", a);
    }
    printf("Данные записаны в файл test.txt\n");
    fclose(f);
    return 0;
}
```

Функции для форматированного ввода и вывода: **fprintf()**, **fscanf()**.

Эти функции схожи с аналогичными функциями, работающими с клавиатурой и консолью, но предназначены для работы с файлами.

Пример. Записать в текстовый файл с именем **test.txt**, который находится в текущей папке проекта, числа от 0 до 100, кратные 3.

В программе **"w"** означает, что файл открывается для записи и, если он существует, то содержимое файла стирается.

Если файл не открывается, то в программе выдается сообщение.

Тип **errno_t** возвращает в соответствующей переменной код возникающей при работе с файлом ошибки.

Функция **perror()** интерпретирует значение глобальной переменной ERRNO в строку и выводит эту строку на стандартный поток вывода с сообщением, указанным в параметре функции.

Макрос EXIT_FAILURE используется для возвращения кода неудачного завершения программы.

2. Изучить *строковый* ввод и вывод при работе с *текстовыми* файлами на языке С, выполнив программу, записанную в данном пункте.

Дополнить программу операторами чтения файла и вывода на экран его содержимого.

```
#include <iostream>
#include <stdio.h>
int main ()
{  setlocale(LC_ALL,"");
   FILE *mf; char str[50];   errno_t err;
   char *estr;
   printf("Введите текст ");   gets(str);
   fopen_s (&mf, "tst.txt", "w");        //Открытие для записи
   fputs(str, mf); fputs("\n", mf);
   fclose (mf);
   err = fopen_s(&mf, "tst.txt", "r");  //Открытие для чтения
   if (err != NULL)                    //Проверка открытия файла
   {   printf ("Ошибка открытия файла\n"); return -1;
   }
   else
   printf ("\nСчитаны строки:\n");
   estr = fgets (str, sizeof(str), mf); //Чтение строки файла
   if (estr == NULL)          //Конец файла или ошибка чтения?
      if (feof(mf) != 0)
          printf ("\nЧтение файла закончено\n");
      else
      {   printf ("\nОшибка чтения из файла\n");
          return -1;
      }
   puts (str);
   if ( fclose (mf) == EOF)
     printf ("Ошибка закрытия\n");
   return 0;
}
```

Функции для строкового ввода и вывода: **fgets()** – чтение из файла одной строки текста полностью, если ее длина меньше 50 символов, **fputs()** – запись строки в файл (функция, в отличие от **puts()** сама не помещает в конце строки '**\n**').

Пример. Записать в текстовый файл с именем **tst.txt** строку символов и прочитать ее.

3. Выполнить программу, записанную в данном пункте, которая осуществляет запись *блока* информации в файл.

Дополнить программу операторами чтения из файла и вывода на экран его содержимого.

С помощью функций **fwrite()** и **fread()** осуществляется блоковый ввод/вывод и работа с бинарными файлами.

Для записи информации служит функция:

fwrite(void *ptr, int size, int n, **FILE *pfile);**

где ***ptr** – указатель на область памяти, в которой размещаются считываемые данные; **size** – размер одного считываемого элемента; **n** – количество считываемых элементов.

Для считывания информации используется функция, общий вид которой:

fread(void *ptr, int size, int n, FILE *pfile);

```
#include <iostream>
#include <stdio.h>
int main(void)
{     setlocale(LC_CTYPE, "Russian");
      FILE *fp;
      errno_t err;
      char const *st = "привет";
      err = fopen_s(&fp,"a.bin", "w");
      if (err != 0)
      {
            perror("ошибка открытия a.txt");
            return EXIT_FAILURE;
      }
      fwrite(st, strlen(st), 1, fp);
       printf("Записан элемент\n");
      fclose(fp);
      return 0;
}
```

В случае успешного считывания информации функция возвращает число прочитанных элементов.

Пример программы на языке **C**, которая открывает файл **a.bin** в текущей папке проекта (если файла нет, то создает его), записывает блок информации, состоящий из строки символов (слово «привет») и закрывает файл.

Функция **fwrite** записывает содержимое переменной **st** длиной **strlen(st)**, **fp** – указатель на файл, в который производится запись.

4. Выполнить программу, записанную в правой части, которая реализует операции *позиционирования* в файле и *блоковый* вывод.

Изменить программу так, чтобы выводились на экран подряд все предложения с 1 по **n**.

Функция **long ftell(FILE *fp)** возвращает **текущую позицию** в файле. Если текущая позиция не определена, функция возвращает -1L.

Функция **int fseek(FILE *fp, long pos, int mode)** устанавливает текущую позицию в файле на байт с номером **pos**. Возвращает 0 при успешном позиционировании и **-1 (EOF)** - в случае неудачи. Параметр **mode** определяет, относительно чего отсчитывается текущая позиция в файле (0 – относительно начала файла, 1 – относительно текущей позиции, 2 – относительно конца файла).

Пример. Пусть в текстовом файле “**b.bin**” (в текущей папке проекта) записано несколько предложений. Конец предложения обозначен точкой. Требуется ввести номер предложения **number** и вывести из текстового файла на экран предложение с этим номером.

```
#include <iostream>
using namespace std;
void main()
{  setlocale(LC_CTYPE, "Russian");
   int m = 0, number = 0, pr = 0;
   long fsize;
   char pd;
   FILE *fd;
   errno_t err;
   err = fopen_s(&fd,"b.txt", "r");
   if (err != 0)
   {    perror("ошибка открытия a.txt");
        return;
   }
   fseek(fd, 0L, SEEK_END);
   fsize = ftell(fd);
   fseek(fd, 0L, SEEK_SET);
```

```
    printf("Введите номер предложения ");
    scanf("%d", &number);
    for (int k = 1; k <= fsize; k++)
    {    fread((void*)&pd, sizeof(char), 1, fd);
         if (pd == '.')   pr++;
         if ((number - 1) == pr)  m++;
         if (number == pr)
         {    long pos1 = k - m - 1;
              if(number != 1) pos1++;
              fseek(fd, pos1, SEEK_SET);
              printf("%d-е предложение: ", number);
              for (int i = 0; i <= m; i++)
              {    fread((void*)&pd, sizeof(char), 1, fd);
                   printf("%c", pd);
              }
              break;
         }
    }
    if (number > pr)
        printf("Такого номера нет");
    fclose(fd);
}
```

Здесь в программе открывается файл, с помощью функции **fseek** устанавливается текущая позиция в файле на конец, и функцией **ftell** определяется размер содержимого файла.

Далее с помощью функции **fseek** устанавливается текущая позиция в файле на начало и происходит чтение информации функцией **fread** и поиск точки.

5. В соответствии со своим вариантом разработать программы для условий, приведенных в таблице ниже, и изучить способы работы с файлами на языке С.

№ варианта	Условие задачи
1	1. Даны два файла вещественных чисел с именами **fA** и **fB**, содержащие элементы прямоугольных матриц **M1** и **M2** по строкам, причем начальный элемент каждого файла содержит количество столбцов соответствующей матрицы. Создать файл той же структуры с именем **fC**, содержащий произведение матриц **M1·M2**. Если матрицы **M1** и **M2** нельзя перемножать, то оставить файл **fC** пустым. 2. Компоненты файла **f** – целые двухзначные числа (положительные и отрицательные). Получить файл **g**, образованный из **f** включением только чисел кратных **K**.
2	1. Даны два файла вещественных чисел с именами **fA** и **fB**, содержащие элементы прямоугольных матриц **M1** и **M2** по строкам, причем начальный элемент каждого файла содержит количество столбцов соответствующей матрицы. Создать файл той же структуры с именем **fC**, содержащий сумму **M1+M2**. 2. Компоненты файла **f** – целые двухзначные (отличные от нуля) числа, причем 10 положительных чисел, 10 отрицательных, и т. д. Получить файл **g**, в котором записаны сначала 5 положительных чисел, затем 5 отрицательных и т. д.
3	1. Дан файл вещественных чисел, содержащий элементы квадратной матрицы (по строкам), причем начальный элемент файла содержит значение количества столбцов матрицы. Создать новый файл той же структуры, содержащий транспонированную матрицу. 2. Даны три файла целых чисел одинакового размера с именами **NameA**, **NameB** и **NameC**. Создать новый файл с именем **NameD**, в котором чередовались бы элементы исходных файлов с одним и тем же номером: **A0, B0, C0, A1, B1, C1, A2, B2, C2, ...**

№ варианта	Условие задачи
4	1. Компоненты файла **fileA** – целые отличные от нуля числа, причем положительных чисел столько же, сколько отрицательных. Получить файл **fileB**, в котором не было бы двух соседних чисел с одинаковым знаком (сохранить порядок следования чисел). 2. Компоненты файла **f** – целые двухзначные числа. Получить файл **g**, образованный из **f** включением только чисел больших некоторого числа, вводимого с клавиатуры.
5	1. Даны два файла целых чисел, содержащие элементы квадратных матриц с именами **A** и **B** по строкам, причем начальный элемент каждого файла содержит количество столбцов соответствующей матрицы. Создать файл той же структуры с именем **C**, содержащий произведение матриц **A** и **B**. 2. Создать текстовый файл **F1** не менее, чем из 6 строк, и записать в него информацию. Скопировать в файл **F2** только те строки из **F1**, которые начинаются с буквы «**A**».
6	1. Компоненты файла **fA** – вещественные числа (положительные и отрицательные). Определить и вывести на экран порядковый номер того из них, которое наиболее близко к введенному пользователем целому числу. 2. Создать текстовый файл **F1** не менее, чем из 6 строк, и записать в него информацию. Скопировать в файл **F2** только четные строки из **F1**.
7	1. Даны три файла целых чисел одинакового размера с именами **NameA**, **NameB** и **NameC**. Создать новый файл с именем **NameD**, в который записать максимальные элементы исходных файлов с одним и тем же номером: **max(*A0, B0, C0)***, **max(*A1, B1, C1)***, **max(*A2, B2, C2)*, ...** 2. Даны три файла целых чисел одинакового размера с именами **A**, **B** и **C**. Создать новый файл с именем **D**, в котором чередовались бы элементы исходных файлов с одним и тем же номером: **a0, b0, c0, a1, b1, c1, a2, b2, c2,** ... .

№ варианта	Условие задачи
8	1. Дан файл целых чисел с элементами **A(i), i = 0, ..., N – 1** (**N** – размер файла). Заменить исходное расположение его элементов на следующее: **A(0), A(N – 1), A(1), A(N – 2), A(2), ...** 2. Даны два файла целых чисел, содержащие элементы матрицы **A** размерности **n** x **n** и **B** размерности **n** x **1** по строкам, причем начальный элемент каждого файла содержит количество столбцов соответствующей матрицы. Создать файл той же структуры с именем **C**, содержащий произведение матриц **A** и **B**.
9	1. Компоненты файла **fileA** – целые числа, значения которых повторяются. Получить файл **fileB**, образованный из **fileA** исключением повторных вхождений одного и того же числа. 2. Создать текстовый файл **F1** не менее, чем из 4 строк, и записать в него информацию. Скопировать из файла **F1** в файл **F2** строки, количество символов в которых больше, чем заданное число.
10	1. Компоненты файла **fileA** – целые отличные от нуля числа: **x, y1,... yn**. Вывести на экран два члена этой последовательности, среднее арифметическое которых ближе всего к **x**. 2. Создать текстовый файл **F1** не менее, чем из 5 строк, и записать в него информацию. Скопировать в файл **F2** только строки из **F1**, которые не содержат цифр.
11	1. Компоненты файла **fileA** – целые числа, значения которых повторяются. Получить файл **fileB**, образованный из **fileA** числами, которые встречаются в **A** ровно 2 раза. 2. Создать текстовый файл **F1** не менее, чем из 6 строк, и записать в него информацию. Скопировать в файл **F2** только те строки из **F1**, которые заканчиваются символом «**a**».
12	1. Дан файл вещественных чисел, содержащий элементы квадратной матрицы по строкам, причем начальный элемент файла содержит значение количества столбцов матрицы. Создать новый файл той

№ варианта	Условие задачи
	же структуры, содержащий **k**-ую строку исходной матрицы. 2. Даны три файла целых чисел одинакового размера с именами **NameA**, **NameB** и **NameC**. Создать новый файл с именем **NameD**, в который записать минимальные элементы исходных файлов с одним и тем же номером: **min(*A0, B0, C0)***, **min(*A1, B1, C1)***, **min(*A2, B2, C2)*, ...**
13	1. Компоненты файла **fileA** – целые числа, значения которых повторяются. Получить файл **fileB**, образованный из **fileA** числами, которые встречаются в **fileA** более 2 раз. 2. Компоненты файла **f** – целые числа. Получить файл **g**, образованный из **f** исключением повторных вхождений одного и того же числа.
14	1. Дан файл вещественных чисел, содержащий элементы квадратной матрицы по строкам, причем начальный элемент файла содержит значение количества столбцов матрицы. Создать новый файл той же структуры, содержащий **k**-ый столбец исходной матрицы. 2. Создать текстовый файл **F1** не менее, чем из 5 строк, и записать в него информацию. Скопировать в файл **F2** только строки из **F1**, которые начинаются с цифры.
15	1. Дан файл вещественных чисел, содержащий элементы квадратной матрицы по строкам, причем начальный элемент файла содержит значение количества столбцов матрицы. Создать новый файл той же структуры, содержащий **k**-ый столбец исходной матрицы. 2. Создать текстовый файл **F1** не менее, чем из 8 строк, и записать в него информацию. Скопировать из файла **F1** в файл **F2** строки, начиная с **k** до **k + 3**.
16	1. Даны два файла целых чисел с именами **fileA** и **fileB**. Получить новый файл с именем **fileC**, который содержит сумму элементов файлов **fileA** и **fileB**. 2. Компоненты файла **file1** – целые двухзначные (отличные от нуля) числа, причем сначала записа-

№ варианта	Условие задачи
	ны 5 положительных чисел, затем 5 отрицательных, и т. д. Получить файл **file2**, в котором записаны числа из файла **file1**, сначала 10 положительных чисел, затем 10 отрицательных и т. д.

Тест "Работа с файлами на языке С"

В начало практикума

Лабораторная работа № 3. Работа с файлами на языке С++

Для обработки файлов на языке С++ в программу должны быть включены заголовочные файлы <**iostream**> и <**fstream**>. Заголовочный файл <**fstream**> включает определения классов потоков: **ifstream** (для ввода в файл), **ofstream** (для вывода из файла), **fstream** (для ввода/вывода). Следует также использовать стандартное пространство имен **– using namespace std;**

Файловый ввод/вывод организован с помощью операций включения (<<) и извлечения (>>).

Задание	Краткие теоретические сведения
1. Изучить запись текста в файл и его чтение на языке С++, выполнив программу, записанную справа.	Пример. Создать текстовый файл с именем **t.txt** и записать в него строку «Работа с файлами в С++». Затем эту информацию прочитать и вывести в консольное окно.

```
#include <fstream>
#include <iostream>
using namespace std;
void main()
{   setlocale(LC_ALL, "rus");
    char buff[50];            // буфер для хранения считываемого из файла текста
    ofstream fout("t.txt");   // создание объекта fout класса ofstream для записи
    fout << "Работа с файлами в С++"; // запись строки в файл
    fout.close();
    ifstream fin("t.txt");    // создание объекта fin класса ifstream для чтения
    if (!fin.is_open())
        cout << "Файл не может быть открыт!\n";
    else
    {   fin >> buff;           // считывание первого слова из файла
        cout << buff << endl;  // печать слова
        fin.getline(buff, 50); // считывание строки из файла
```

```
        fin.close();
        cout << buff << endl;   // печать строки
    }
}
```

В программе показаны два способа чтения из файла, первый – используя операцию передачи в поток, второй – используя функцию **getline()**. В первом случае считывается только первое слово, во втором случае считывается строка длиной 50 символов. Так как в файле осталось меньше 50 символов, то считываются символы до последнего включительно. Считывание во второй раз продолжилось после первого слова, а не с начала, так как первое слово было прочитано ранее.

2. В правой части приведена программа с использованием *функций пользователя*, которая открывает файл с записанным в нем числом (имя файла с расширением нужно ввести с клавиатуры), читает из файла *число*, возводит его в квадрат и записывает в файл.

Выполнить программу, используя различные имена файлов.

```
#include <iostream>
#include <fstream>
using namespace std;
double inFile(ifstream &f, char s[]); //Функция чтения из файла
void fromFile(ofstream &f, double a, char s[]); //Функция записи в файл

void main()
{    setlocale(LC_ALL, "Russian");
     double a, b;
     char str[40];
     ifstream ifile;
     ofstream ofile;
     cout << "\n Ввести имя файла для чтения: \n";
     cin >> str;
     a = inFile(ifile, str);
     cout << "\n Прочитанное из файла число = " << a;
     b = pow(a, 2);
     cout << "\n b=" << b;
```

```
        cout << "\n Ввести имя файла для записи: \n";
        cin >> str;
        fromFile(ofile, b, str);
        cout << endl;
    }
    double inFile(ifstream &f, char s[40]) //Функция чтения из файла
    {
        double a;
        f.open(s);
        if (f.fail())  //проверка открытия файла
        {    cout << "\n Ошибка открытия файла";
             exit(1);
        }
        f >> a;      //чтение числа из файла в переменную a
        f.close();
         return a;
    }
    void fromFile(ofstream &f, double a, char s[40]) //Функция записи в файл
    {    f.open(s);
        if (f.fail())
        {    cout << "\n Ошибка открытия файла";
             exit(1);
        }
        f << a;            //запись числа из переменной a в файл
        f.close();
    }
```

Здесь имена потоков для чтения **ifile** и для записи **ofile** передаются из главной функции в подпрограммы как параметры.

3. Выполнить программу, записанную в правой части, которая осуществляет запись *блока* информации в файл и его считывание на языке С++.

С помощью функций **fwrite()** и **fread()** осуществляется блоковый ввод/вывод и работа с бинарными файлами.

Для записи информации на языке С++ служит функция:

fwrite(void *ptr, int size);

где ***ptr** – указатель на область памяти, в которой размещаются считываемые данные; **size** – размер одного считываемого элемента.

```
#include <iostream>
#include <fstream>
using namespace std;
void main(void)
{   setlocale(0, "Rus");
    char const *st = "привет";  char *buf;
    ofstream fout("acplus.bin");
    if (fout.fail())
    {   cout << "\n Ошибка открытия файла";
        exit(1);
    }
    fout.write((char *)&st, sizeof(st));
    cout<<"Записан элемент\n";
    fout.close();
    ifstream fin("acplus.bin");
    if (fin.fail())
    {   cout << "\n Ошибка открытия файла";
        exit(1);
    }
    fin.read((char *)&buf, sizeof(st));
    cout<<buf<<endl;
    fin.close();
}
```

Для считывания информации на языке С++ используется функция, общий вид которой:

fread(void *ptr, int size);

В случае успешного считывания информации функция возвращает число прочитанных элементов.

Пример программы на языке **С**, которая открывает файл **acplus.bin** в текущей папке проекта (если файла нет, то создает его), записывает блок информации, состоящий из строки символов (слово «привет») и закрывает файл. Затем открывает файл и считывает блок информации.

4. В программе справа представлено совместное использование способов работы с файлами на языке С и на языке С++.

Пример. Пусть дано число **k** и строковый файл на диске **f** с именем **name1**, содержащий непустые строки. Создать программным путем два новых файла: строковый с именем **name2**, содержащий последние **k** символов каждой строки исходного файла (если строка короче **k** символов, то она сохраняется целиком), и символьный с именем **name3**, содержащий **k**-й символ каждой строки (если строка короче **k**, то в файл **name3** записывается пробел).

```
#include <stdio.h>
#include <fstream>
using namespace std;
int main()
{      int k; FILE *fin;
       fopen_s(&fin,"f:\\name1.txt", "rt");
       ofstream fout1("f:\\name2.txt");
       ofstream fout2("f:\\name3.txt");
       printf("Vvedite 4islo k\n");
       scanf_s("%d", &k);
       while (!feof(fin))
       {     char s[255] = "";
             fgets(s, 254, fin);
             if (strlen(s) <= k)
                   fout1 << s;
             else
             {     for (int i = strlen(s) - k - 1; i < strlen(s); i++)
                         fout1 << s[i];
             }
             if (strlen(s) < k)
                   fout2 << " " << endl;
             else
                   fout2 << s[k - 1] << endl;
       }
```

```
        fclose(fin);
        fout1.close();
        fout2.close();
        return 0;
    }
```

5. В соответствии со своим вариантом разработать программы для работы с файлами на языке **C++**. Для первой программы необходимо предварительно создать текстовый файл **FILE1** из нескольких строк и записать в него данные. Во второй программе ввод информации с клавиатуры и вывод в консольное окно должно осуществляться в главной функции, а запись в файл и чтение из файла – в функциях пользователя. Встроенные функции для работы со строками не использовать.

№ варианта	**Условие задачи**
1	1. Скопировать в файл **FILE2** только четные строки из **FILE1**. Подсчитать размер файлов **FILE1** и **FILE2** (в байтах). 2. Ввести с клавиатуры строку символов, состоящую из цифр и слов, разделенных пробелами, и записать ее в файл. Прочитать из файла данные и вывести нечетные числа строки.
2	1. Скопировать в файл **FILE2** только те строки из **FILE1**, которые начинаются с буквы «**A**». Подсчитать количество слов в **FILE2**. 2. Ввести с клавиатуры строку символов, состоящую из групп цифр и нулей, и записать ее в файл. Прочитать из файла данные и вывести самую короткую группу.
3	1. Скопировать из файла **FILE1** в файл **FILE2** строки, начиная с **к** до **к+3**. Подсчитать количество гласных букв в **FILE2**. 2. Ввести с клавиатуры строку символов, состоящую из цифр и слов, разделенных пробелами, и записать ее в файл. Прочитать из файла данные, вывести слова строки и записать их в другой файл.

№ варианта	Условие задачи
4	1. Скопировать из файла **FILE1** в файл **FILE2** все строки, которые не содержат цифры. Подсчитать количество строк, которые начинаются на букву «**A**» в файле **FILE2**. 2. Ввести с клавиатуры строку символов, состоящую из цифр и слов, разделенных пробелами, и записать ее в файл. Прочитать из файла данные, подсчитать количество символов в самом длинном слове и вывести его.
5	1. Скопировать из файла **FILE1** в файл **FILE2** строки, начиная с четвертой по порядку. Подсчитать количество символов в последнем слове **FILE2**. 2. Ввести с клавиатуры строку, состоящую из букв, цифр, запятых, точек, знаков + и –, и записать ее в файл. Прочитать из файла данные и вывести подстроку, которая соответствует записи целого числа.
6	1. Скопировать из файла **FILE1** в файл **FILE2** строки, начиная с **N** до **K**. Подсчитать количество согласных букв в файле **FILE2**. 2. Ввести с клавиатуры строку символов, состоящую из групп цифр и нулей, и записать ее в файл. Прочитать из файла данные и вывести на экран группы с четным количеством символов.
7	1. Скопировать из файла **FILE1** в файл **FILE2** все строки, которые содержат только одно слово. Найти слово, содержащее 5 символов, в файле **FILE2**. 2. Ввести с клавиатуры строку символов, состоящую из цифр и слов, разделенных пробелами, и записать ее в файл. Прочитать из файла данные и вывести на экран номер слова, содержащего **k**-й по счету с начала символ. Если в **k**-й позиции пробел, то вывести номер предыдущего слова.
8	1. Скопировать из файла **FILE1** в файл **FILE2** все строки, кроме той строки, в которой больше всего гласных букв. Напечатать номер этой строки.

№ варианта	Условие задачи
	2. Ввести с клавиатуры строку символов, состоящую из букв, цифр, запятых, точек, знаков + и –, и записать ее в файл. Прочитать из файла данные и вывести подстроку, которая соответствует записи вещественного числа с фиксированной точкой.
9	1. Скопировать из файла **FILE1** в файл FI**L**E2 все строки, начинающиеся на букву «**C**», расположенные между строками с номерами **N**1 и **N2**. Определить количество слов в первой строке файла **FILE2**. 2. Ввести с клавиатуры строку символов, состоящую из слов, разделенных пробелами, и записать ее в файл. Прочитать из файла данные и вывести на экран порядковый номер слова минимальной длины. Посчитать количество символов в слове.
10	1. Скопировать из файла **FILE1** в файл **FILE2** все строки, в которых нет слов, совпадающих с первым словом. Определить количество согласных букв в первой строке файла **FILE2**. 2. Ввести с клавиатуры строку символов, состоящую из слов, разделенных пробелами, и записать ее в файл. Прочитать из файла данные и вывести все слова, которые содержат букву «**р**».
11	1. Скопировать из файла **FILE1** в файл **FILE2** все строки, которые содержат только одно слово. Найти самое короткое слово в файле **FILE2**. 2. Ввести с клавиатуры две строки символов, состоящих из слов, разделенных пробелами, и записать их в файл. Прочитать из файла данные. Найти самое короткое слово в первой строке и самое длинное во второй строке. Посчитать количество гласных букв в этих словах.
12	1. Скопировать из файла **FILE1** в файл **FILE2** все строки, в которых есть слова, совпадающие с первым словом. Определить количество согласных букв в последней строке файла **FILE2**. 2. Ввести с клавиатуры строку символов, состоящую из букв, цифр, запятых, точек, знаков + и –, и записать ее в файл. Прочитать из файла данные и посчитать и вывести количество запятых.

№ варианта	Условие задачи
13	1. Скопировать из файла **FILE1** в файл **FILE2** все строки, в которых более 2 слов. Определить номер слова, в котором больше всего гласных букв. 2. Ввести с клавиатуры строку символов, состоящую из цифр и скобок, и записать ее в файл. Прочитать из файла данные, посчитать и вывести количество скобок различного вида.
14	1. Скопировать из файла **FILE1** в файл FI**L**E2 все строки, в которых содержится два одинаковых слова. Определить номер слова, в котором больше всего букв «**1**». 2. Ввести с клавиатуры строку символов, состоящую из слов, разделенных пробелами, и записать ее в файл. Прочитать из файла данные. Разбить строку на две подстроки, причем первая длиной **k** символов. Если на **k**-ю позицию попадает слово, то его следует отнести ко второй строке.
15	1. Скопировать из файла **FILE1** в файл **FILE2** все строки, в которых содержится не менее двух одинаковых слов. Определить номер слова, в котором больше всего цифр. 2. Ввести с клавиатуры строку символов, состоящую из слов, разделенных пробелами, и записать ее в файл. Прочитать из файла данные и вывести все слова, которые содержат букву «**x**».
16	1. Скопировать из файла **FILE1** в файл **FILE2** все строки, в которых есть слова, совпадающие со вторым словом. Определить количество цифр в последней строке файла **F2**. 2. Ввести с клавиатуры строку символов, состоящую из цифр и слов, разделенных пробелами, и записать ее в файл. Прочитать из файла данные и вывести четные числа строки.

Тест "Работа с файлами на языке С++"

В начало практикума

Лабораторная работа № 4. Представление информации в виде структуры

Структура – это пользовательский тип данных, совокупность логически связанных данных различного типа, объединенных под одним идентификатором.

Задание	Краткие теоретические сведения
1. Изучить способы организации данных в виде *структуры*, выполнив программу, записанную в данном пункте. Дополнить структуру дополнительными сведениями о работниках и вывести их вместе с фамилиями на экран.	Пример. Пусть надо описать структуру с именем `Worker`, содержащую следующие поля: фамилия работника; название занимаемой должности; год поступления на работу. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в массив, состоящий из трех объектов структуры `Worker`; вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры; если таких работников нет, то на дисплей должен выводиться соответствующий текст. В программе структура описана следующим образом: `struct Worker` `{ char lastName[30];` `char position[30];` `int year;` `};` Здесь предусмотрены компоненты: **`lastName[30]`** для ввода фамилий, **`position[30]`** – для должности, **`year`** – для года поступления на работу. Запись `Worker` **`a[size]`** означает, что **`a[size]`**– массив из объектов структуры в количестве **`[size]`.** Для доступа к членам структуры используется *операция принадлежности* “**`.`**” (точка): **`a[i].lastName`**, **`a[i].position`**, **`a[i].year`**.

```cpp
#include <iostream>
using namespace std;

struct Worker
{   char lastName[30];
    char position[30];
    int year;
};

int main()
{   setlocale(LC_ALL, "Russian");
    const int size = 3;
    const int currentYear = 2017;
    int i, b, counter = 0;
    Worker a[size];
    for (i = 0; i < size; i++)
    {     cout << "Введите фамилию " << i + 1 << "-ого работника " << endl;
          cin >> a[i].lastName;
          cout << endl << "Должность: ";
          cin >> a[i].position;
          cout<< endl << "Год поступления на работу: ";
          cin >> a[i].year;
    }
    cout << endl << "Введите стаж работы ";
    cin >> b;
    cout << endl;
    for (i = 0; i < size; i++)
    {     if (b < currentYear - a[i].year)
          {     cout << a[i].lastName << " ";
                counter++;
```

```
            }
            else
                  if (i == size-1 && counter > 0)
                        cout << "Работников с более высоким стажем нет.";
    }
    return 0;
}
```

2. В правой части приведен пример программы, в которой используется *указатель на структуру*.

Выполнить программу, меняя скорость вывода информации.

Написать пояснения к программе.

При обращении к структуре с помощью указателя используется *операция косвенной адресации* -> (стрелка).

Программа осуществляет отображение на экране программного таймера пользователя. Скорость работы таймера можно регулировать, изменяя макроопределение **DELAY**.

```
#include <conio.h>
#include <iostream>
using namespace std;
#define DELAY 12800
void update(struct MyTime *t);
void display(struct MyTime *t);

struct MyTime
{   int hours;
    int minutes;
    int seconds;
}

int main(void)
{
    struct MyTime St;
```

```
    St.hours = 0;
    St.minutes = 0;
    St.seconds = 0;
    for (; ;)
    {    update(&St);
         display(&St);
         if (_kbhit())
              return 0;
    }
}

void update(struct MyTime *t)
{
     t->seconds++;
     if (t->seconds == 60)    { t->seconds = 0; t->minutes++; }
     if (t->minutes == 60)    { t->minutes = 0; t->hours++; }
     if (t->hours == 24)
          t->hours = 0;
     for (long int i = 1; i < DELAY; ++i);
}

void display(struct MyTime *t)
{
      cout << t->hours << '.' << t->minutes << '.' << t->seconds << endl;
}
```

Функция **_kbhit()** возвращает истину, если нажата какая-либо клавиша на клавиатуре. В противном случае возвращается 0.

3. В правой части приведен пример программы, которая осуществляет работу с данными, организованными в виде структуры, содержащей информацию о студентах: №, Фамилия, Год рождения, Факультет.

Проанализировать работу главной функции и функций, входящих в программу.

Добавить операторы выдачи сообщений об ошибках при вводе данных неправильного типа.

```
#include "stdafx.h"
#include <iostream>
# define str_len 30
# define size 30
using namespace std;

void enter_new();
void del();
void change();
void out();

struct Student
{     char name[str_len];
      int year_of_birth;
      char department[5];
};
struct Student list_of_student[size];
struct Student bad;
int current_size = 0; int choice;

int main()
{     setlocale(LC_CTYPE, "Russian");
      cout << "Введите:" << endl;
      cout << "1-для удаления записи" << endl;
      cout << "2-для ввода новой записи" << endl;
      cout << "3-для изменения записи" << endl;
      cout << "4-для вывода записи(ей)" << endl;
      cout << "5-для выхода" << endl;
      cin >> choice;
      do
      {     switch (choice)
```

```
                {     case 1:  del(); break;
                      case 2:  enter_new();  break;
                      case 3:  change();  break;
                      case 4:  out(); break;
                }
          } while (choice != 5);
     }
     void enter_new()
     {     cout << "Ввод информации" << endl;
          if (current_size < size)
          {     cout << "Строка номер ";
                cout << current_size + 1;
                cout << endl << "Фамилия " << endl;
                cin >> list_of_student[current_size].name;
                cout << "Год рождения " << endl;
                cin >> list_of_student[current_size].year_of_birth;
                cout << "Факультет " << endl;
                cin >> list_of_student[current_size].department;
                current_size++;
          }
          else
                cout << "Введено максимальное кол-во строк";
          cout << "Что дальше?" << endl;
          cin >> choice;
     }
     void del()
     {     int d;
          cout << "\nНомер строки, которую надо удалить (для удаления всех строк нажать
     99)" << endl;  cin >> d;
          if (d != 99)
```

```
            {     for (int de1 = (d - 1); de1 < current_size; de1++)
                        list_of_student[de1] = list_of_student[de1 + 1];
                  current_size = current_size - 1;
            }
            if (d == 99)
                  for (int i = 0; i < size; i++)
                        list_of_student[i] = bad;
            cout << "Что дальше?" << endl;
            cin >> choice;
      }
      void change()
      {     int n, per;
            cout << "\nВведите номер строки" << endl;  cin >> n;
            do
            {     cout << "Введите: " << endl;
                  cout << "1-для изменения фамилии" << endl;
                  cout << "2-для изменения года рождения" << endl;
                  cout << "3-для изменения факультета" << endl;
                  cout << "4-конец\n";
                  cin >> per;
                  switch (per)
                  {     case 1: cout << "Новая фамилия";
                              cin >> list_of_student[n - 1].name;   break;
                        case 2: cout << "Новый год рождения";
                              cin >> list_of_student[n - 1].year_of_birth; break;
                        case 3: cout << "Новый факультет ";
                              cin >> list_of_student[n - 1].department; break;
                  }
            } while (per != 4);
            cout << "Что дальше?" << endl;
```

```
        cin >> choice;
    }
    void out()
    {   int sw, n;
        cout << "1-вывод 1 строки" << endl;
        cout << "2-вывод всех строк" << endl;
        cin >> sw;
        if (sw == 1)
        {   cout << "Номер выводимой строки " << endl;   cin >> n;  cout << endl;
            cout << "Фамилия ";
            cout << list_of_student[n - 1].name << endl;
            cout << "Год рождения ";
            cout << list_of_student[n - 1].year_of_birth << endl;
            cout << "Факультет ";
            cout << list_of_student[n - 1].department << endl;
        }
        if (sw == 2)
        {   for (int i = 0; i < current_size; i++)
            {   cout << "Фамилия ";
                cout << list_of_student[i].name << endl;
                cout << "Год рождения ";
                cout << list_of_student[i].year_of_birth << endl;
                cout << "Факультет ";
                cout << list_of_student[i].department << endl;
            }
        }
        cout << "Что дальше?" << endl;
        cin >> choice;
    }
```

4. Выполнить программму, приведенную в правой части, несколько раз с различными исходными данными.

Убедиться, что данные, введенные с клавиатуры, записываются в файл с именем **base.bin**.

Записать условие задачи и комментарии к программе.

```
#include <iostream>
using namespace std;

void input(int size);
void output();
void find(char lastName[]);

typedef struct Students
{     char fio[16];
      char group[3];
} STUD;
int number; FILE *f; errno_t err;

int main()
{     setlocale(LC_ALL, "Russian");
      int choice; char fio[16];
      do
      {     cout << "\n1.Ввод данных с клавиатуры и запись в файл\n";
            cout << "2.Вывод данных из файла\n";
            cout << "3.Поиск по фамилии\n";
            cout << "0.Выход из программы\n\n";
            cout << "Введите номер операции: ";
            cin >> choice;
            switch (choice)
            {
            case 1: cout << "Введите количество студентов: ";
                    cin >> number;
                    input(number);  break;
            case 2: output(); break;
            case 3: {  cout << "Введите фамилию: ";
                       cin >> fio;
```

```
                        find(fio); break;
                    }
            case 0: exit(0);  break;
            }
      } while (choice != 0);
}

void input(int size)
{     STUD buf = { ' ', ' ' };
      if (!fopen_s(&f, "base.bin", "ab"))
      {     for (int p = 0; p < size; p++)
            {     cout << "Фамилия: ";  cin >> buf.fio;
                  cout << "Группа: ";   cin >> buf.group;
                  fwrite(&buf, sizeof(buf), 1, f);
            }
            fclose(f);
      }
      else {   cout << "Ошибка открытия файла";
               return;
            }
}

 void output()
{     STUD buf;
      if (!fopen_s(&f, "base.bin", "rb"))
      {     cout << "\nФамилия    Группа\n";
            fread(&buf, sizeof(buf), 1, f);
            while (!feof(f))
            {  cout << buf.fio << "\t    " << buf.group << endl;
               fread(&buf, sizeof(buf), 1, f);
```

```
            }
            cout << endl;
            fclose(f);
      }
      else
      {     cout << "Ошибка открытия файла";
            return;
      }
}

void find(char lastName[16])
{     bool flag = false;  STUD buf;
      if (!fopen_s(&f, "base.bin", "rb"))
      {     while (!feof(f))
            {     fread(&buf, sizeof(buf), 1, f);
                  if (strcmp(lastName, buf.fio) == 0)   //сравнение строк
                  {     cout << "\nФамилия    Группа\n";
                        cout << buf.fio << "\t    "<< buf.group << endl;
                        flag = true; break;
                  }
            }
            fclose(f);
            if (!flag) cout << "Ничего не найдено\n";
      }
      else
      {     cout << "Ошибка открытия файла";
            return;
      }
}
```

4. В соответствии со своим вариантом разработать программу для данных, приведенных в таблице ниже. Определить структурированный тип, разработать меню для работы с массивом структур.

В программу должны войти функции:

- ввод элементов структуры с клавиатуры;
- вывод элементов структуры в консольное окно;
- удаление заданной структурированной переменной;
- поиск информации;
- запись информации в файл;
- чтение данных из файла.

№ варианта	Условие задачи
1	**Горожанин.** Ф.И.О., дата рождения, адрес, пол (м, ж). Реализовать выборку по году рождения.
2	**Список клиентов гостиницы.** Паспортные данные, даты приезда и отъезда, номер, тип размещения (люкс, одноместный, двухместный, трехместный, апартаменты). Поиск гостя по фамилии.
3	**Клиенты банка**. Ф.И.О., тип счета (срочный, льготный и т. д.), номер счета, сумма на счете, дата последнего изменения. Выбор по номеру счета.
4	**Личная библиотека.** Автор книги, название, издательство, раздел библиотеки (специальная литература, хобби, домашнее хозяйство, беллетристика и т. д.), происхождение (покупка, кража, подарок) и наличие книги в данный момент. Выбор книг по автору.
5	**Ломбард.** База хранимых товаров и недвижимости: анкетные данные клиента, наименование товара, оценочная стоимость; сумма, выданная под залог, дата сдачи, срок хранения. Выбор по наименованию товара.

№ варианта	Условие задачи
6	**Склад.** Наименование товара, цена, количество, процент торговой надбавки (5, 10, 15, 20, 35, 30). Выбор по цене.
7	**Авиарейсы**. Номер рейса, пункт назначения, время вылета, дата вылета, стоимость билета, количество мест. Выбор по пункту назначения.
8	**Ученики**. Ф.И.О., класс (цифра+буква) предметы, оценки, средний балл. Выбор по среднему баллу.
9	**Студенты**. Ф.И.О., дата поступления, специальность, группа, факультет, средний балл. Выбор по фамилии.
10	**Справочник автомобилей.** Марка автомобиля, цвет, заводской номер, дата выпуска, тип кузова (седан, универсал и т. п.), дата последнего техосмотра, владелец. Выбор транспортных средств по владельцу.
11	**Записная книжка.** Ф.И.О, дата рождения, адрес, телефон, место работы или учебы, должность. Автоматическое формирование поздравления с днем рождения (по текущей дате).
12	**Государство**. Наименование, столица, численность населения, площадь, фамилия президента. Выбор государства по названию.
13	**Вокзал**. Номер поезда, пункт назначения, дни следования, время прибытия, время отправления. Выбор по пункту назначения.
14	**Справочник абитуриента.** Наименование вуза, адрес, перечень специальностей, конкурс прошлого года по каждой специальности, размер оплаты при договорном обучении. Выбор по специальности.
15	**Преподаватели**. Фамилия преподавателя, название экзамена, дата экзамена. Выбор по фамилии.
16	**Отдел кадров**. Ф.И.О. работника, образование, специальность, подразделение, должность, оклад, дата поступления на предприятие. Выбор по должности.

5. Дополнительные задания

1. Определить структуру для представления информации о сданных студентом экзаменах, содержащую поля: ФИО студента, число экзаменов, полученные оценки. Определить функции для обработки отдельного объекта (например, для проверки, сданы ли все экзамены на 4 и 5). Написать функцию для обработки массива структур. В результате обработки требуется вычислить характеристику успеваемости студентов, то есть отношение числа студентов, сдавших экзамены на 4 и 5, к общему числу студентов, в процентах.

2. Описать структуру с именем TRAIN, содержащую поля: названия пункта назначения, номер поезда, время отправления. Написать программу, выполняющую ввод с клавиатуры данных в массив, состоящий из восьми элементов типа TRAIN (записи должны быть размещены в алфавитном порядке по названиям пунктов назначения); вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени (если таких поездов нет, то вывести сообщение об этом).

3. Определить структуру для представления информации о наличии санаторных путевок, содержащую следующие поля: Название санатория, Место расположения, Лечебный профиль, Количество путевок. Представить введенные данные в виде таблицы, сгруппировав их по лечебным профилям санаториев. В пределах каждой группы данные отсортировать по названиям санаториев. Организовать поиск информации и вывод результатов.

Тест "Представление информации в виде структуры"

В начало практикума

Лабораторная работа № 5. Объединения, перечисления, битовые поля

Задание	Краткие теоретические сведения
1. Изучить принципы работы с *объединениями*, выполнив программу, записанную в данном пункте.	*Объединение* – это поименованная совокупность данных разных типов, размещаемых в *одной и той же области памяти*, размер которой достаточен для хранения наибольшего элемента. Объединение подобно структуре, однако в каждый момент времени может использоваться *только один* из элементов объединения. В примере программы выводятся компоненты объединения **Utypes.** Первая часть программы выполняется без ошибок, поскольку данные разных типов присваиваются объединению не одновременно, а последовательно. Во второй части правильно отобразится лишь значение типа **double**, поскольку оно было занесено последним.

```
#include <iostream>
using namespace std;

union Utypes
{   char c;
    int ivalue;
    float fvalue;
    double dvalue;
} mun;

int main(void)
{   setlocale(LC_ALL, "Russian");
    mun.c = 'b';
    cout << mun.c << endl;
    mun.ivalue = 1990;
```

```
    cout << mun.ivalue << endl;
    mun.fvalue = 19.90;
    cout << mun.fvalue << endl;
    mun.dvalue = 987654.32E+13;
    cout << mun.dvalue << endl;

    // ниже приведены операторы некорректного вывода
    cout << mun.c << endl;
    cout << mun.ivalue << endl;
    cout << mun.fvalue << endl;
    cout << mun.dvalue << endl;
    cout << "Размер объединения составляет "<< sizeof(Utypes) << " байт.";
    return 0;
}
```

2. Изучить принципы работы с *перечислениями*, выполнив программу, записанную в данном пункте.

Перечисления используются в тех случаях, когда данные можно представить в виде нумерованного списка.

В примере создается перечисление `Emonths`, включающее названия месяцев.

Здесь перечисление представляет собой ряд целых чисел от 1 до 12. Например, значение переменной **months**, после того как ей была присвоена константа **December**, стало равным 12.

Поскольку названию каждого месяца соответствует определенное числовое значение, то элементы перечисления могут участвовать в арифметических операциях**.**

```
#include <iostream>
using namespace std;
enum Emonths
{   January = 1, February, March,
    April, May, June, July, August, September,
    October, November, December
} months;
```

```
int main(void)
{   setlocale(LC_CTYPE, "Russian");
    int current_month, left;
    cout <<"Введите номер текущего месяца (от 1 до 12):";
    cin >> current_month;
    months = December;
    left = (int)months - current_month;
    cout << "\nДо конца года осталось "<< left << " месяца(ев)\n";
    return 0;
}
```

3. Изучить принципы работы с *битовыми полями*, выполнив программу, записанную в данном пункте.

Элементом структуры может быть *битовое поле,* обеспечивающее доступ к отдельным битам памяти, которые позволяют рационально использовать память с помощью хранения данных в минимально требуемом количестве битов. Элементы битового поля должны быть объявлены как тип **int** или **unsigned**.

В программе отображается в двоичной системе счисления ASCII-код, который генерируется при нажатии любой клавиши. При этом используется объединение **bits** и битовые поля **byte**. Объединение позволяет присвоить значение нажатой клавиши символьной переменной, а битовые поля используются для отображения отдельных битов.

Программа прекращает работу при нажатии буквы **q**.

```
#include <iostream>
using namespace std;
struct byte
{   unsigned a : 1;
    unsigned b : 1;
    unsigned c : 1;
    unsigned d : 1;
```

```
    unsigned e : 1;
    unsigned f : 1;
    unsigned g : 1;
    unsigned h : 1;
};
union bits
{   char ch;
    struct byte bit;
} ascii;

void disp_bits(bits b);

void main()
{   do
    {     cin >> ascii.ch;
          disp_bits(ascii);
    } while (ascii.ch != 'q');   //выход при вводе q
}

void disp_bits(bits b)
{   if (b.bit.h) cout << "1";  else cout << "0";
    if (b.bit.g) cout << "1";  else cout << "0";
    if (b.bit.f) cout << "1";  else cout << "0 ";
    if (b.bit.e) cout << "1";  else cout << "0";
    if (b.bit.d) cout << "1";  else cout << "0";
    if (b.bit.c) cout << "1";  else cout << "0";
    if (b.bit.b) cout << "1";  else cout << "0";
    if (b.bit.a) cout << "1";  else cout << "0";
    cout << "\n";
}
```

4. В соответствии со своим вариантом разработать программу с использованием *перечислений* и *битовых полей* для работы с данными из таблицы, приведенной ниже. Реализовать функции ввода с клавиатуры, вывода на экран, удаления, поиска элементов. Интерфейс пользователя осуществить в виде меню.

№ варианта	Условие задачи
1	**Отдел кадров**. Ф.И.О. работника, образование, специальность, подразделение, должность, оклад, дата поступления на предприятие. Выбор по стражу работы. Даты реализовать с помощью битового поля, должность – с помощью перечисления.
2	**Горожанин.** Ф.И.О., дата рождения, адрес, пол (м, ж). Выборка по году рождения. Дату рождения организовать с помощью битового поля, пол – с помощью перечисления.
3	**Ученики**. Ф.И.О., класс (цифра+буква) предметы, оценки, средний балл. Выбор по фамилии. Класс реализовать с помощью битового поля, предметы – через перечисление.
4	**Клиенты банка**. Ф.И.О., тип счета (срочный, льготный и т. д.), номер счета, сумма на счете, дата последнего изменения. Выбор по диапазону суммы (<100, >100). Дату реализовать с помощью битового поля, тип счета – с помощью перечисления.
5	**Личная библиотека.** Автор книги, название, издательство, раздел библиотеки (специальная литература, хобби, домашнее хозяйство, беллетристика и т. д.), происхождение (покупка, кража, подарок) и наличие книги в данный момент. Выбор книг по году. Происхождение книги реализовать с помощью перечисления.
6	**Справочник автомобилей.** Марка автомобиля, цвет, заводской номер, дата выпуска, тип кузова (седан, универсал и т. п.), дата последнего техосмотра, владелец. Выбор транспортных средств по номеру. Дату выпуска реализовать с помощью битового поля, марку – с помощью перечисления.

№ варианта	Условие задачи
7	**Склад.** Наименование товара, цена, количество, процент торговой надбавки (5, 7, 11, 20, 25, 30). Выбор по цене. Процент торговой надбавки реализовать с помощью перечисления.
8	**Авиарейсы**. Номер рейса, пункт назначения, время вылета, дата вылета, стоимость билета, количество мест. Выбор по дате вылета. Дату вылета реализовать с помощью битового поля, пункт назначения – с помощью перечисления.
9	**Вокзал**. Номер поезда, пункт назначения, дни следования, время прибытия, время отправления. Выбор по пункту назначения. Дни следования реализовать с помощью перечисления. Время выбытия и прибытия реализовать с помощью битового поля (часы, минуты).
10	**Государство**. Наименование, столица, численность населения, площадь. Выбор государства по занимаемой площади (> заданного значения). Форму правления реализовать с помощью перечисления.
11	**Ломбард.** Фамилия клиента, наименование товара, оценочная стоимость, сумма, выданная под залог, дата сдачи, срок хранения. Выбор товаров по истечении срока хранения. Дату сдачи реализовать с помощью битового поля.
12	**Записная книжка.** Ф.И.О, дата рождения, адрес, телефон, место работы или учебы, должность. Поиск по фамилии. Дату рождения реализовать с помощью битового поля.
13	**Студенты**. Ф.И.О., дата поступления, специальность, группа, факультет, средний балл. Выбор по среднему баллу. Дату поступления реализовать с помощью битового поля, факультет – с помощью перечисления.
14	**Список клиентов гостиницы.** Паспортные данные, даты приезда и отъезда, номер, тип размещения (люкс, одноместный, двухместный, трехместный, апартаменты). Поиск гостя по дате приезда. Даты приезда и отъезда реализовать с помощью битового поля, тип размещения – с помощью перечисления.

№ варианта	Условие задачи
15	**Справочник абитуриента.** Наименование вуза, адрес, перечень специальностей, конкурс прошлого года по каждой специальности, размер оплаты при договорном обучении. Выбор по минимальному конкурсу. Конкурс прошлого года по каждой специальности реализовать через битовые поля, перечень специальностей – через перечисления.
16	**Преподаватели**. Фамилия преподавателя, название экзамена, дата экзамена. Выбор по дате экзамена. Дату экзамена реализовать с помощью битового поля.

5. В соответствии со своим вариантом разработать программу с использованием структуры в виде *объединения,* для работы с данными из таблицы, приведенной ниже. Реализовать функции ввода с клавиатуры, вывода на экран и поиска.

№ варианта	Условие задачи
1	**Преподаватели**. Фамилия преподавателя, название экзамена, дата экзамена. Выбор по фамилии.
2	**Вокзал**. Номер поезда, пункт назначения, дни следования, время прибытия, время отправления. Выбор по пункту назначения.
3	**Ломбард.** База хранимых товаров и недвижимости: анкетные данные клиента, наименование товара, оценочная стоимость; сумма, выданная под залог, дата сдачи, срок хранения. Выбор товаров по наименованию.
4	**Клиенты банка**. Ф.И.О., тип счета (срочный, льготный и т. д.), номер счета, сумма на счете, дата последнего изменения. Выбор по номеру счета.
5	**Личная библиотека.** Автор книги, название, издательство, раздел библиотеки (специальная литература,

№ варианта	Условие задачи
	хобби, домашнее хозяйство, беллетристика и т. д.), происхождение (покупка, кража, подарок) и наличие книги в данный момент. Выбор книг по автору.
6	**Отдел кадров**. Ф.И.О. работника, образование, специальность, подразделение, должность, оклад, дата поступления на предприятие. Выбор по должности.
7	**Склад.** Наименование товара, цена, количество, процент торговой надбавки (5, 10, 15, 20, 35, 30). Выбор по наименованию.
8	**Авиарейсы**. Номер рейса, пункт назначения, время вылета, дата вылета, стоимость билета, количество мест. Выбор по пункту назначения.
9	**Ученики**. Ф.И.О., класс (цифра+буква) предметы, оценки, средний балл. Выбор по фамилии.
10	**Горожанин.** Ф.И.О., дата рождения, адрес, пол (м, ж). Реализовать выборку по году рождения.
11	**Справочник автомобилей.** Марка автомобиля, цвет, заводской номер, дата выпуска, тип кузова (седан, универсал и т. п.), дата последнего техосмотра, владелец. Выбор транспортных средств по владельцу.
12	**Записная книжка.** Ф.И.О, дата рождения, адрес, телефон, место работы или учебы, должность. Поиск по фамилии.
13	**Государство**. Наименование, столица, численность населения, площадь, фамилия президента. Выбор государства по названию.
14	**Список клиентов гостиницы.** Паспортные данные, даты приезда и отъезда, номер, тип размещения (люкс, одноместный, двухместный, трехместный, апартаменты). Поиск гостя по фамилии.
15	**Справочник абитуриента.** Наименование вуза, адрес, перечень специальностей, конкурс прошлого года

№ варианта	Условие задачи
	по каждой специальности, размер оплаты при договорном обучении. Поиск минимального конкурса по данной специальности.
16	**Студенты**. Ф.И.О., дата поступления, специальность, группа, факультет, средний балл. Выбор по среднему баллу.

Тест "Объединения, битовые поля, перечисления"

В начало практикума

Лабораторная работа № 6. Динамические структуры данных. Односвязные списки

Список представляет собой линейную последовательность переменных, каждая из которых связана указателями со своими соседями. Списки бывают *односвязные* (каждый элемент имеет указатель на следующий); *двусвязные* (каждый элемент имеет указатель на следующий и на предыдущий); *двусвязные циклические* (первый и последний элементы ссылаются друг на друга).

Задание	Краткие теоретические сведения
1. Изучить работу с *односвязным списком*, выполнив программу, представленную в правой части. Написать условие задачи.	Пример реализации односвязного списка с помощью массива структур. `#include <iostream>` `using namespace std;` `struct Item` `{  int info;` `  Item* next;` `};` `void main()` `{  Item *plist = nullptr;  //указатель на начало списка` `  Item *p;  int number;` `  for (;;)  // создание списка` `  {  cout << "Input number ";` `  cin >> number;  //ввод чисел, пока не введен 0` `  if (!number)  break;` `  p = new Item;` `  p->info = number;` `  p->next = plist;` `  plist = p;` `  }` `  p = plist;`

```
    while (p)                    // перебор списка с выводом элементов
    {     cout << p->info << ' ';
          p = p->next;
     }
    while (plist)                // перебор списка и удаление элементов
    {     p = plist;
          plist = fplist->next;
          delete p;
     }
}
```

Ключевое слово **nullptr** используется для обнуления указателей.

2. В правой части приведена программа, реализующая работу с односвязным списком с использованием *функций пользователя*.

Выполнив программу, изучить способы передачи параметров в функции.

Пример. Создать список, содержащий числа вещественного типа. Найти сумму элементов больших 7.

```
#include <iostream>
using namespace std;
struct list
{   float number;
    list *next;
};

void insert(list *&, float); //функция добавления элемента, передается адрес спис-
ка и символ, который добавляется
float del(list *&, float);   //функция удаления, передается адрес списка и символ,
который удаляется
int IsEmpty(list *);         //функция, которая проверяет, пуст ли список
void printList(list *);      //функция вывода
void menu(void);     //функция, показывающая меню
void sum7(list *);  //функция подсчета суммы чисел, больших 7
```

```
int main()
{   setlocale(LC_CTYPE, "Russian");
    list *first = NULL;
    int choice;
    float value;
    menu();    // вывести меню
    cout<<" ? ";
    cin>>choice;
    while (choice != 4)
    {   switch (choice)
        {    case 1:    cout<<"Введите число "; // добавить число в список
                        cin>>value;
                        insert(first, value);
                        printList(first);
                        break;
             case 2:    if (!IsEmpty(first)) // удалить число из списка
                        {    cout<<"Введите удаляемое число ";
                             cin>>value;
                             if (del(first, value))
                             {    cout<<"Удалено число " << value<<endl;
                                  printList(first);
                             }
                             else
                                  cout<<"Число не найдено"<<endl;
                        }
                        else
                             cout<<"Список пуст"<<endl;
                        break;
             case 3:    sum7(first);    // вычисление суммы
```

```
                        break;
                default:  cout<<"Неправильный выбор"<<endl;
                          menu();
                          break;
            }
           cout<<"?  ";
           cin>>choice;
     }
    cout<<"Конец"<<endl;
    return 0;
}

void menu(void)  //Вывод меню
{   cout<<"Сделайте выбор:"<<endl;
    cout<<" 1 - Ввод числа"<<endl;
    cout<<" 2 - Удаление числа"<<endl;
    cout<<" 3 - Вычисление суммы чисел, больших 7"<<endl;
    cout<<" 4 - Выход"<<endl;
}

void insert(list *&p, float value) //Добавление числа value в список
{  list *newP = new list;
   if (newP!= NULL)     //есть ли место?
   {    newP->number = value;
        newP->next = p;
        p = newP;
   }
   else
        cout<<"Операция добавления не выполнена"<<endl;
}
```

```
float del(list *&p, float value)  // Удаление числа
{   list *previous, *current, *temp;
    if (value == p->number)
    {    temp = p;
         p = p->next;    // отсоединить узел
         delete temp;      //освободить отсоединенный узел
         return value;
    }
    else
    {    previous = p;
         current = p->next;
         while (current!= NULL && current->number != value)
         {   previous = current;
             current = current->next; // перейти к следующему
         }
         if (current!= NULL)
         {    temp = current;
              previous->next = current->next;
              free(temp);
              return value;
         }
    }
    return 0;
}

int IsEmpty(list *p) //Список  пустой? (1-да, 0-нет)
{    return p == NULL;   }
```

```
void printList(list *p)  //Вывод списка
{    if (p == NULL)
         cout<<"Список пуст"<<endl;
     else
     {   cout<<"Список:"<<endl;
         while (p!= NULL)
         {   cout<<"-->"<<p->number;
             p = p->next;
         }
         cout<<"-->NULL"<<endl;
     }
}

void sum7(list *p)  // Подсчет суммы чисел, больших 7
{    float sm = 0;
     if (p == NULL)
           cout<<"Список пуст"<<endl;
     else
     {    while (p!= NULL)
          {    if(p->number>7)
                    sm = sm + (p->number);
               p = p->next;
          }
            cout<<"Сумма = "<<sm<<endl;
     }
}
```

3. В правой части приведена программа, реализующая работу с односвязным списком.

Выполнив программу, изучить способы *записи списка в файл* и *считывания* из файла.

Пример. Создать список, содержащий символы. Реализовать функции записи списка в файл и считывания списка из файла.

```
#include <iostream>
#include <fstream>
using namespace std;

void insert(list *&p, char value); //Добавление символа в начало списка
void printList(list *p);       //Вывод списка
void toFile(list *&p);         //Запись в файл
void fromFile(list *&p);       //Считывание из файла
void menu(void);               //Вывод меню

struct list
{   char symbol;
    list *next;
};

int main()
{   setlocale(LC_CTYPE, "Russian");
    list *first =nullptr;
    int choice; char value;
    menu();    // вывести меню
    cout<<" ? ";
    cin>>choice;
    while (choice != 4)
    {   switch (choice)
       {       case 1:   cout<<"Введите символ ";
                         cin>>value;
                         insert(first, value);
                         printList(first);
```

```
                                    break;
                    case 2:    toFile(first);
                                    break;
                    case 3:    fromFile(first);
                                    break;
                    default:   cout<<"Неправильный выбор"<<endl;
                                    menu(); break;
            }
             cout<<"?  ";
             cin>>choice;
    }
    return 0;
}

void insert(list *&p, char value) //Добавление символа в начало списка
{   list *newP = new list;
    if (newP!= NULL)     //есть ли место?
    {    newP->symbol = value;
         newP->next = p;
         p = newP;
    }
    else
         cout<<"Операция добавления не выполнена"<<endl;
}

void printList(list *p)  //Вывод списка
{    if (p == NULL)
         cout<<"Список пуст"<<endl;
     else
     {   cout<<"Список:"<<endl;
```

```
            while (p!= NULL)
            {   cout<<"-->"<<p->symbol;
                p = p->next;
            }
            cout<<"-->NULL"<<endl;
        }
}

void toFile(list *&p)
{   list *temp = p;
    list buf;
    ofstream frm("mList.dat");
    if (frm.fail())
    {   cout << "\n Ошибка открытия файла";
       exit(1);
    }
    while (temp)
    {  buf = *temp;
       frm.write((char *)&buf, sizeof(list));
       temp = temp->next;
   }
   frm.close();
   cout << "Список записан в файл mList.dat\n";
}

void fromFile(list *&p)          //Считывание из файла
{  list buf, *first = nullptr;
   ifstream frm("mList.dat");
   if (frm.fail())
   {    cout << "\n Ошибка открытия файла";
```

```
            exit(1);
        }
        frm.read((char *)&buf, sizeof(list));
        while (!frm.eof())
        {   insert(first, buf.symbol);
            cout<<"-->"<<buf.symbol;
            frm.read((char *)&buf, sizeof(list));
        }
        cout<<"-->NULL"<<endl;
        frm.close();
        p = first;
        cout << "\nСписок считан из файла mList.dat\n";
}

void menu(void)     //Вывод меню
{   cout<<"Сделайте выбор:"<<endl;
    cout<<" 1 - Ввод символа"<<endl;
    cout<<" 2 - Запись списка в файл"<<endl;
    cout<<" 3 - Чтение списка из файла"<<endl;
    cout<<" 4 - Выход"<<endl;
}
```

4. В соответствии со своим вариантом разработать программу с использованием *односвязного списка* по данным, представленным в таблице ниже.

Программа должна содержать меню с пунктами: добавление элемента, удаление элемента, поиск элемента, вывод списка в консольное окно, запись списка в файл, считывание списка из файла.

№ варианта	Условие задачи
1	Создать список, содержащий элементы целого типа. Найти сумму положительных элементов или выдать сообщение, что положительных элементов нет.
2	Создать список, содержащий элементы вещественного типа. Найти среднее значение положительных элементов.
3	Создать список, содержащий элементы целого типа. Найти сумму положительных элементов, кратных 5, или выдать сообщение, что таких элементов нет.
4	Создать список, содержащий символы. Найти символ, равный введенному с клавиатуры, вывести его и следующий за ним символ.
5	Создать список, содержащий элементы целого типа. Найти сумму положительных двухзначных элементов или выдать сообщение, что таких элементов нет.
6	Создать список, содержащий элементы целого типа. Найти сумму отрицательных двухзначных элементов или выдать сообщение, что таких элементов нет.
7	Создать список, содержащий элементы целого типа. Найти сумму отрицательных элементов, у которых последняя цифра 3 или выдать сообщение, что таких элементов нет.
8	Создать список, содержащий элементы целого типа. Найти сумму отрицательных элементов, кратных 2, или выдать сообщение, что таких элементов нет.
9	Создать список, содержащий элементы вещественного типа. Найти среднее значение отрицательных элементов.
10	Создать список, содержащий элементы вещественного типа. Найти произведение элементов, значение каждого из которых меньше 10.

11	Создать список, содержащий символы. Найти символ, равный введенному с клавиатуры, вывести этот символ и предыдущий.
12	Создать список, содержащий элементы вещественного типа. Найти среднее значение положительных элементов.
13	Создать список, содержащий элементы целого типа. Найти сумму отрицательных двухзначных элементов или выдать сообщение, что таких элементов нет.
14	Создать список, содержащий элементы целого типа. Найти сумму отрицательных элементов, кратных 5, или выдать сообщение, что таких элементов нет.
15	Создать список, содержащий элементы целого типа. Найти сумму положительных элементов, у которых последняя цифра 7 или выдать сообщение, что таких элементов нет.
16	Создать список, содержащий элементы вещественного типа. Найти сумму элементов, каждый из которых меньше числа 9.

Контрольная работа № 1

В начало практикума

Лабораторная работа № 7. Полустатические структуры данных: стеки

Стеком называется одномерная структура данных, включение и исключение элементов которого осуществляется с помощью указателя стека в соответствии с правилом "последним введен, первым выведен" (last-in, first-out – **LIFO**). Стек Стек может быть реализован статически (на основе *массива*) и динамически (на основе *списка*).

Задание	Краткие теоретические сведения
1. Изучить реализацию стека на основе *динамического массива*, выполнив программу, приведенную в правой части.	Пример. Разработать калькулятор целых чисел на основе стека.

```
#include <iostream>
#define MAX 100
int *p;                 // указатель на область свободной памяти
int *tos, *bos;         // указатель на вершину и дно стека
void push(int i);       //прототип
int pop(void);          //прототип

void main(void)
{  setlocale(LC_CTYPE, "Russian");
   int a, b;  char s[80];
   p = new int[MAX*sizeof(int)];
   if (!p)
   {       printf("Ошибка при выделении памяти\n");
           exit(1);
   }
   tos = p; bos = p + MAX - 1;
   printf("Калькулятор \n Для выхода нажать 'q'\n");
   do
   {    printf(": ");  gets_s(s); //ввод первого числа, второго и знака операции
        switch (*s)
```

```
            {     case '+':  a = pop(); b = pop();   //сложение
                             printf("%d\n", a + b);
                             push(a + b); break;
                  case '-':  a = pop(); b = pop();   //вычитание
                             printf("%d\n", b - a);
                             push(b - a);  break;
                  case '*':  a = pop(); b = pop();   //умножение
                             printf("%d\n", b * a);
                             push(b * a);  break;
                  case '/':  a = pop(); b = pop();   //деление
                             if (a == 0) { printf("Деление на 0\n"); break; }
                             printf("%d\n", b / a);
                             push(b / a);  break;
                  case '.':  a = pop(); push(a); //вывод вершины стека
                             printf("Текущее значение на вершине стека: %d\n", a);
                             break;
                  default:   push(atoi(s)); //конвертация из символа в число
            }
      } while (*s != 'q');
}
void push(int i)     // Занесение элемента в стек
{   if (p > bos) { printf("Стек полон\n"); return; }
      *p = i;  p++;
}
int pop(void)        // Получение верхнего элемента из стека
{   p--;
      if (p < tos) { printf("Стек пуст\n"); return 0; }
      return *p;
 }
```

2. В программе, приведенной справа, демонстрируется реализация стека на основе *односвязного списка*.

```
#include <iostream>
using namespace std;

int pop(stack* &st); // Извлечение элемента
void push(stack* &st, int d); // Добавление элемента

struct stack
{    int data;
     stack *next;
};
int main()
{    stack *p = nullptr;
     push(p, 100);    //число 100 – в стек
     push(p, 200);    //число 200 – в стек
     pop(p);          //вывод текущего элемента = 200
     pop(p);          // вывод текущего элемента = 100
     return 0;
}
int pop(stack* &st) // Извлечение элемента
{    int tmp = st->data;   stack *pv = st;
     st = st->next;       // вершиной становится предшествующий элемент
     delete pv;           // освобождается память
     cout << tmp << endl; //вывод текущего элемента
     return tmp;
}
void push(stack* &st, int d) // Добавление элемента
{    stack *pv = new stack;
     pv->data = d;   // значение помещается в стек
     pv->next = st;
     st = pv;
}
```

3. В правой части представлен *проект, состоящий из трех частей*: программный модуль с главной функцией, программный модуль с функциями пользователя, осуществляющими операции со стеком, и заголовочный файл.

В программе создается стек из целых чисел от 0 до 9 на основе *односвязного списка*.

Первая часть – программный модуль с главной функцией (пусть он имеет имя **proectStack**) – организует вывод меню и обращения к функциям работы со стеком, описанным в файле **myStack.cpp**.

Для записи главной функции используется обычная процедура создания проекта. Надо после открытия **MS Visual Studio** выполнить **Создать проект** / **Visual C++** / **Win32** / **Консольное приложение Win32**.

В появившемся окне в поле **Имя** следует ввести имя проекта (**proectStack**), указать место размещения проекта.

В окне **Мастер приложений Win32** нажать кнопку **Далее**, в появившемся окне нажать **Готово**. В глобальной области появится заготовка:

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}
```

В окне кода надо записать текст программы.

```
// proectStack.cpp: определяет точку входа для консольного приложения.
   #include "stdafx.h"
   #include <iostream>
   #include "myStack.h"
   using namespace std;
   int _tmain(int argc, _TCHAR* argv[])
   {    setlocale(LC_ALL, "Rus");
        int choice;
        Stack *myStk = new Stack; //выделение памяти для стека
        myStk->head = NULL;       //инициализация первого элемента
        for (;;)
```

```
        {     cout << "Выберите команду:" << endl;
              cout << "1 - Добавление элемента в стек" << endl;
              cout << "2 - Извлечение элемента из стека" << endl;
              cout << "3 - Вывод стека" << endl;
              cout << "4 - Выход" << endl;
              cin >> choice;
              switch (choice)
              {     case 1: cout << "Введите элемент: " << endl;
                            cin >> choice;
                            push(choice, myStk);
                            break;
                    case 2: choice = pop(myStk);
                            if (choice != -1)
                                cout << "Извлеченный элемент: " << choice << endl;
                            break;
                    case 3: cout << "Весь стек: " << endl;
                            show(myStk);
                            break;
                    case 4: return 0;
                            break;
              }
        }
        return 0;
    }
```

Для добавления *второго* программного модуля с именем **myStack.cpp**, содержащего функции для манипуляции со стеком (добавление, удаление и вывод элементов), надо в контекстном меню пункта **Файлы исходного кода** окна **Обозревателя решений** выполнить **Добавить** / **Создать элемент**. В появившемся окне выбрать **Код** / **Файл C++**, набрать имя (**myStack.cpp**) и ввести тексты функций.

```
#include "stdafx.h"
#include <iostream>
#include "myStack.h"
using namespace std;
void push(int x, Stack *myStk)   //Добавление элемента x в стек
{  Stack* e = new Stack;    //выделение памяти для нового элемента
   e->data = x;             //запись элемента x в поле v
   e->next = myStk->head;   //перенос вершины на следующий элемент
   myStk->head = e;         //сдвиг вершины на позицию вперед
}
int pop(Stack *myStk)   //Извлечение (удаление) элемента из стека
{  if (myStk->head == NULL)
   {    cout << "Стек пуст!" << endl;
        return -1;              //если стек пуст - возврат -1
   }
   else
   {    Stack *e = myStk->head;   //e - переменная для хранения адреса элемента
        int a = myStk->head->data;   //запись числа из поля data в переменную a
        myStk->head = myStk->head->next; //перенос вершины
        delete e;                        //удаление временной переменной
        return a;                        //возврат значения удаляемого элемента
   }
}
void show(Stack *myStk)    //Вывод стека
{  Stack* e = myStk->head;    //объявляется указатель на вершину стека
   int a;
   if (e == NULL)
        cout << "Стек пуст!" << endl;
   while (e != NULL)
```

```
    {     a = e->data;          //запись значения в переменную a
          cout << a << " ";
          e = e->next;
    }
    cout << endl;
}
```

Третья часть – это заголовочный файл **myStack.h**, который содержит описание структуры стека и прототипы программ обработки его элементов.

Этот файл добавляется в заголовочную часть. Надо в контекстном меню пункта **Заголовочные файлы** окна **Обозревателя решений** выполнить **Добавить** / **Создать элемент**. В появившемся окне выбрать **Код** / **Заголовочный файл**, набрать имя (**myStack.h**) и ввести текст программы.

```
struct Stack
{   int data;             //информационный элемент
    Stack *head;          //вершина стека
    Stack *next;          //указатель на следующий элемент
};

void show(Stack *myStk);          //прототип
int pop(Stack *myStk);            //прототип
void push(int x, Stack *myStk);   //прототип
```

4. Создать проект, демонстрирующий работу со стеком, организованным на основе *списка*, в соответствии со своим вариантом для данных из таблицы ниже. Все операции со стеком реализовать через функции. Дополнить проект функциями очистки стека **clear()**, сохранения в файл и считывания из файла.

Проект должен содержать три части: главная функция, файл с функциями работы со стеком и заголовочный файл. Создать интерфейс в виде меню.

№ варианта	Задание для выполнения
1	Разработать функцию, которая по одному стеку строит два новых: **Stack1** из положительных элементов и **Stack2** из отрицательных.
2	Разработать функцию, которая удаляет из стека первый отрицательный элемент, если такой есть.
3	Разработать функцию, которая удаляет первый положительный элемент, если такой есть.
4	Разработать функцию, которая определяет, есть ли в стеке хотя бы один элемент, который равен следующему за ним элементу.
5	Разработать функцию, которая формирует стек **Stack**, включив в него по одному разу элементы, которые входят в стек **Stack1**, но не входят в стек **Stack2**.
6	Разработать функцию, которая подсчитывает количество элементов стека, у которых равные "соседи".
7	Разработать функцию, которая удаляет из стека первый элемент, значение которого превышает число 100, если такой элемент есть.
8	Разработать функцию, которая по одному стеку строит два новых: **Stack1** из элементов, значение которых превышает число 50, и **Stack2** – из остальных элементов.
9	Разработать функцию, которая формирует стек **Stack**, включив в него по одному разу элементы, которые входят в один из стеков **Stack1** и **Stack2**, но не входят в другой.
10	Разработать функцию, которая определяет, есть ли в стеке хотя бы один элемент, лежащий в заданном диапазоне.

№ варианта	Задание для выполнения
11	Разработать функцию удаления элементов стека, кратных 3, если такие есть.
12	Разработать функцию, которая по одному стеку **Stack** строит 2 новых: **Stack1** из четных элементов, **Stack2** из нечетных.
13	Разработать функцию подсчета количества повторяющихся элементов стека.
14	Разработать функцию, которая формирует стек **Stack**, включив в него повторяющиеся элементы стеков **Stack1** и **Stack2**.
15	Разработать функцию, которая удаляет первый повторяющийся элемент стека.
16	Разработать функцию, которая удаляет из стека **Stack1** элементы, входящие в стек **Stack2**, но не входящие в стек **Stack1**, и наоборот.

5. Дополнительные задания.

1. Дана величина *a* строкового типа из четного количества символов. Получить и напечатать величину *b*, состоящую из символов первой половины величины *a*, записанных в обратном порядке, после которых идут символы второй половины величины *a*, также записанные в обратном порядке (например, при *a* = “привет” *b* должно быть равно «ипртев»).

2. Создать стек с целочисленным информационным полем. Заполнить его *длинами строк*, считанных из файла. Распечатать на экране содержимое стека. Указать номер и длину последней самой короткой строки файла.

Тест "Стек"

В начало практикума

Лабораторная работа № 8. Полустатические структуры данных: очереди

Очередь – одномерная структура данных, для которых загрузка или извлечение элементов осуществляется с помощью указателей начала извлечения (**head**) и конца (**tail**) очереди в соответствии с правилом **FIFO** ("**first-in, first-out**" – "первым введен, первым выведен"), т. е. включение производится с одного, а исключение – с другого конца.

Задание	Краткие теоретические сведения
1. В программе, приведенной справа, демонстрируется реализация очереди на основе *односвязного списка.* Внести изменения в программу с тем, чтобы выводились не только буквы, но и слова.	`#include <iostream>` `using namespace std;` `void intoFIFO(Queue *ph[], int v); //Постановка элемента в конец очереди` `void scan(Queue *ph[]);  //Вывод элемента` `char fromFIFO(Queue *ph[]);  //Извлечение элемента` `struct Queue` `{  char symbol;` `  Queue *next;` `};` `void main()` `{  Queue A3 = { 'a', NULL }, A2 = { 'b', &A3 }, A1 = { 'c', &A2 };` `  Queue*  ph[2];` `  ph[0] = &A1;` `  ph[1] = &A3;` `  scan(ph);` `  intoFIFO(ph, 'd');` `  intoFIFO(ph, 't');` `  scan(ph);` `  char vv = fromFIFO(ph);` `  scan(ph);` `}`

```
void intoFIFO(Queue *ph[], int v) //Постановка элемента в конец очереди
{     Queue *p = new Queue;
      p->symbol = v;
      p->next = NULL;
      if (ph[0] == NULL)
            ph[0] = ph[1] = p;    //включение в пустую очередь
      else
      {
            ph[1]->next = p;
            ph[1] = p;
      }
}

void scan(Queue *ph[])            //Вывод элемента
{
      for (Queue* p = ph[0]; p != NULL; p = p->next)
            cout << p->symbol <<' ';
      cout <<endl;
}

char fromFIFO(Queue *ph[])       //Извлечение элемента
{     Queue *q;
      if (ph[0] == NULL) return -1;      // очередь пуста
      q = ph[0];                         // исключение первого элемента
      ph[0] = q->next;
      if (ph[0] == NULL) ph[1] = NULL;
      char v = q->symbol;
      return v;
}
```

2. Изучить способы манипуляции с элементами очереди, реализованной на основе односвязного списка выполнив программу, приведенную в правой части.

Пример. Создать очередь для целых чисел. Максимальный размер очереди ввести с клавиатуры. Создать функции для ввода, вывода и удаления элементов очереди. Найти минимальный элемент в очереди и удалить все элементы до него.

```
#include<iostream>
using namespace std;
struct Number
{   int info;
    Number *next;
};
void create(Number **begin, Number **end, int p); //формирование элементов очереди
void view (Number *begin); //функция вывода элементов очереди
Number *minElem (Number *begin); //функция определения минимального элемента
void DeltoMin (Number **begin, Number **p); //функция удаления до минимального
элемента
int main()
{    Number *begin = NULL, *end, *t;
     t = new Number;
     int p,size;
     cout << "\nEnter size queue=";  cin >> size;
     cout << "Enter number= ";       cin >> p;
     t->info = p;        //первый элемент
     t->next = NULL;
     begin = end = t;
     for(int i = 1; i < size; i++) //создание очереди
     {   cout << "Enter number= ";    cin >> p;
         create(&begin, &end, p);
     }
     cout << "\nelements of queue: \n";
```

```
        if (begin == NULL)   //вывод на экран
            cout << "No elements" << endl;
        else
            view(begin);
        t = minElem (begin);    //определение минимального
        cout<<"minimum="<<t->info<<endl;
        DeltoMin (&begin, &t);   //удаление до минимального
        cout << "\nnew Queue:\n";
        view(begin);
        return 0;
}
void create (Number **begin, Number **end, int p) //Формирование элементов очереди
{   Number *t = new Number;
    t->next = NULL;
    if (*begin == NULL)
       *begin = *end = t;
    else
    {  t->info = p;
       (*end)->next = t;
       *end = t;
    }
}
void view (Number *begin) //Вывод элементов очереди
{   Number *t = begin;
    if (t == NULL)
    {  cout << "Number is empty\n";
       return;
    }
    else
       while (t != NULL)
```

```
            {   cout << t->info << endl;
                t = t->next;
            }
    }
    Number *minElem (Number *begin) //Опрделение минимального элемента
    {   Number *t = begin, *mn;
        int min;
        if (t == NULL)
        {  cout << "Number is empty\n"; return 0; }
        else
        {  min = t->info;
           while (t != NULL)
           {    if(t->info <= min)
                {   min = t->info;
                    mn = t;
                }
                t = t->next;
           }
        }
        return mn;
    }
    void DeltoMin (Number **begin, Number **p) //Удаление до минимального элемента
    {    Number *t;
         t = new Number;
         while (*begin != *p)
         {   t = *begin;
             *begin = (*begin)->next;
              delete t;
         }
    }
```

3. В правой части приведен *проект*, в котором реализация очереди осуществлена на основе *динамического массива*.

Изменить главную функцию, включив операторы работы с функциями добавления, извлечения и вывода различных элементов.

Программный модуль с главной функцией:

```
// ProectQueue.cpp: определяет точку входа для консольного приложения.
//
#include "stdafx.h"
#include "myQueue.h"
#include <iostream>
using namespace std;

struct myQue
{    int a;
     char b;
};
void printQueue(Queue& s)    // Вывод на экран с очисткой очереди
{  while (!(s.isEmpty()))
   {   cout << ((myQue *)peekQueue(s))->a << "  " << ((myQue *)peekQueue(s))->b <<
endl;
       delQueue(s);
   }
}
int _tmain(int argc, _TCHAR* argv[])
{  Queue q1 = createQueue(4);
   myQue a1 = { 1, 'q' }, a2 = { 2, 'w' }, a3 = { 3, 'e' };
   enQueue(q1, &a1);
   enQueue(q1, &a2);
   enQueue(q1, &a3);
   myQue* a4 = new myQue;
   a4->a = 4;
   a4->b = 'r';
   enQueue(q1, a4);
```

```
    printQueue(q1);
    return 0;
}
```

Программный модуль MyQueue.cpp

```
#include "stdafx.h"
#include "myQueue.h"
Queue createQueue(int n)          // Выделить ресурс для очереди
{   return *(new Queue(n));
};
Queue createQueue(const Queue &pq)   // Создать очередь
{   Queue *rc = new Queue(pq.Size - 1);
    rc->Head = pq.Head;
    rc->Tail = pq.Tail;
    for (int i = 0; i < pq.Size; i++)
         rc->Data[i] = pq.Data[i];
    return *rc;
}
bool Queue::isFull() const          // Очередь заполнена?
{     return (Head % Size == (Tail + 1) % Size); }
bool Queue::isEmpty()const          // Очередь пуста?
{     return (Head % Size == Tail % Size); }
bool enQueue(Queue &q, void* x)      // Добавить элемент x
{   bool rc = true;
    if (rc = !q.isFull())
    {     q.Data[q.Tail] = x;
          q.Tail = (q.Tail + 1) % q.Size;
    }
    else
```

```
        rc = false;
    return rc;
};

void* delQueue(Queue &q)                // Удалить элемент
{   void* rc = (void*)MYQUEUE1_EQE;
    if (!q.isEmpty())
    {     rc = q.Data[q.Head];
          q.Head = (q.Head + 1) % q.Size;
    }
    else
          rc = NULL;
    return rc;
}

void* peekQueue(const Queue &q)  // Получить первый элемент очереди
{   void* rc = (void*)MYQUEUE1_EQE;
    if (!q.isEmpty())
          rc = q.Data[q.Head];
    else
          rc = NULL;
    return rc;
}

int  clearQueue(Queue &q)       // Очистить очередь
{   int rc = (q.Tail - q.Head) >= 0 ? q.Tail - q.Head : (q.Size - q.Head + q.Tail +
1);
    q.Tail = q.Head = 0;
    return rc;     // колич. элементов до очистки
}
```

```
void releaseQueue(Queue &q)     // Освободить ресурсы очереди
{  delete[] q.Data;
   q.Size = 1;
   q.Head = q.Tail = 0;
}
```

Заголовочная функция MyQueue.h

```
#define MYQUEUE1_EQE  0x0000  // возврат в случае пустоты очереди

struct Queue                  // Блок управления очередью
{  int Head;             // голова очереди
   int Tail;             // хвост очереди
   int Size;             // размер очереди (макс. колич.+1)
   void** Data;          // хранилище данных очереди
   Queue(int size)       // физический размер очереди
   {    Head = Tail = 0;
        Data = new void*[Size = size + 1];
   }
   bool isFull() const;        // очередь заполнена ?
   bool isEmpty()const;        // очередь пуста ?
};

Queue createQueue(int n);             // n – макс. количество
Queue createQueue(const Queue& pq);   // создать очередь по образцу
bool  enQueue(Queue& q, void* x);     // добавить x
void* delQueue(Queue& q);             // удалить элемент
void* peekQueue(const Queue& q);      // получить первый элемент
int clearQueue(Queue& q);             // очистить очередь
void releaseQueue(Queue& q);          // освободить ресурсы
```

4. Создать проект из нескольких файлов, демонстрирующий работу с очередью. В соответствии со своим вариантом выполнить задание из таблицы, представленной ниже.

Разработать меню и реализовать все операции с очередью через функции. Максимальный размер очереди ввести с клавиатуры.

№ варианта	Задание для выполнения
1	Создать очередь для символов и функции для ввода, вывода и удаления элементов очереди. Ввести эталонный символ. Вводить символы с экрана в очередь до встречи эталонного. После встречи с эталонным, вывести всю очередь, удалить два элемента и посчитать оставшееся количество элементов очереди.
2	Создать очередь для целых чисел и функции для ввода, вывода и удаления элементов очереди. Найти количество элементов между максимальным и минимальным элементами очереди.
3	Создать очередь для целых (положительных и отрицательных) чисел и функции для ввода, вывода, удаления и определения размера очереди. Удалить три первых элемента очереди. Вывести размер оставшейся очереди. Найти максимальный элемент очереди.
4	Создать очередь для символов и функции для ввода, вывода и определения размера очереди. Ввести символы с экрана в очередь. В случае совпадения вводимого символа с последним элементом очереди удалить первых три элемента.
5	Создать очередь для символов и функции для ввода, вывода и удаления элементов очереди. Ввести символы с клавиатуры в очередь. После введения 5-го символа в ответ на каждый вводимый удалять по одному элементу из очереди.
6	Создать очередь для целых чисел и функции для ввода, вывода, удаления и определения размера очереди. Вывести элементы очереди до первого нулевого (включительно). Вывести размер оставшейся очереди. Найти максимальный элемент очереди.

№ варианта	Задание для выполнения
7	Создать три очереди для целых чисел и функции для ввода, вывода и удаления элементов очереди. При добавлении элементов положительные числа надо поместить в первую очередь, отрицательные во вторую, а нулевые в третью.
8	Создать очередь для целых чисел и функции для ввода, вывода, удаления и определения размера очереди. Разработать функцию, которая удаляет из очереди первый отрицательный элемент, если такой есть.
9	Создать очередь для целых чисел и функции для ввода, вывода, удаления и определения размера очереди. Разработать функцию, которая по одной очереди строит две новых: **Queue1** из четных элементов и **Queue2** – из остальных элементов очереди.
10	Разработать функцию, которая будет находить минимальный элемент очереди, и помещать его в новую очередь **Queue1**, затем будет находить максимальный элемент и помещать его в очередь **Queue2**, затем снова минимальный и так до тех пор, пока из одной очереди не образуется две.
11	Создать 2 очереди для символов и функции для ввода, вывода и удаления элементов очереди. Ввести в очередь символы с экрана. Прописные буквы преобразовать в строчные и поместить в первую очередь, строчные буквы преобразовать в прописные и поместить во вторую очередь.
12	Создать очередь для символов и функции для ввода, вывода, удаления и определения размера очереди. Ввести символы с экрана в очередь. В случае совпадения вводимого символа с первым элементом очереди вывести очередь и ее размер.
13	Создать очередь для массива целых (положительных и отрицательных) чисел и функции для ввода, вывода и удаления элементов очереди. При вводе чисел в очередь помещать только отрицательные элементы. Найти минимальный и максимальный элемент очереди. Вывести все элементы очереди и посчитать их количество.

№ варианта	Задание для выполнения
14	Создать очередь для символов и функции для ввода, вывода и удаления элементов очереди. Ввести символы с клавиатуры. Ввести эталонный символ и проверить, имеется ли он в очереди. При его наличии в очереди, удалить все элементы до него.
15	Создать очередь для символов. Создать функции для ввода, вывода, удаления и определения размера очереди. Ввести эталонный символ. Вводить символы с экрана в очередь до встречи эталонного. При встрече эталонного, удалить два элемента очереди. Вывести размер очереди.
16	Создать очередь для чисел. Создать функции для ввода, вывода и удаления элементов очереди. Ввести числа в очередь. Создать две очереди из первой. В одну поместить положительные числа, а во вторую отрицательные.

5. Дополнительное задание.

1. В приложении 1 приведен проект, в котором реализована очередь на основе односвязного *списка* с *приоритетным* включением. На основе данного проекта разработать функции, которые предлагается создать в данном приложении.

2. Создать очередь с вещественными числами, и заполнить ее с клавиатуры. Выполнить циклический сдвиг элементов в очереди так, чтобы в ее начале был расположен наибольший элемент.

3. Содержимое текстового файла **f**, разделенное на строки, переписать в текстовый файл **g**, перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением исходного взаимного порядка, как среди цифр, так и среди остальных литер строки). Использовать очереди.

Тест "Очередь"

В начало практикума

Лабораторная работа № 9. Двусвязные списки

В *двусвязных списках* каждый элемент имеет указатели на следующий и на предыдущий элементы.

Задание	Краткие теоретические сведения
1. В правой части записана программа, которая формирует *двусвязный список* для хранения информации, содержащей адреса людей (имя и город). Написать комментарии к программе.	

```
#include <iostream>
#include <fstream>
using namespace std;
const unsigned int NAME_SIZE = 30;
const unsigned int CITY_SIZE = 20;

struct Address
{     char name[NAME_SIZE];
      char city[CITY_SIZE];
      Address *next;
      Address *prev;
};
//-------------------------------------------------------
int menu(void)
{     char s[80];  int c;
      cout << endl;
      cout << "1. Ввод имени" << endl;
      cout << "2. Удаление имени" << endl;
      cout << "3. Вывод на экран" << endl;
      cout << "4. Поиск" << endl;
      cout << "5. Выход" << endl;
     cout << endl;
      do
```

```
	{	cout << "Ваш выбор: ";
		cin.sync();
		gets_s(s);
		cout << endl;
		c = atoi(s);
	} while (c < 0 || c > 5);
	return c;
}
//-------------------------------------------------------------
void insert(Address *e, Address **phead, Address **plast) //Добавление в конец списка
{	Address *p = *plast;
	if (*plast == NULL)
	{	e->next = NULL;
		e->prev = NULL;
		*plast = e;
		*phead = e;
		return;
	}
	else
	{	p->next = e;
		e->next = NULL;
		e->prev = p;
		*plast = e;
	}
}
 //-------------------------------------------------------------
Address* setElement()		// Создание элемента и ввод его значений с клавиатуры
{	Address* temp = new  Address();
	if (!temp)
	{	cerr << "Ошибка выделения памяти памяти";
```

```
            return NULL;
      }
      cout << "Введите имя: ";
      cin.getline(temp->name, NAME_SIZE-1,'\n');
      cin.ignore(cin.rdbuf()->in_avail());
      cin.clear();
      cout << "Введите город: ";
      cin.getline(temp->city, CITY_SIZE-1,'\n');
      cin.ignore(cin.rdbuf()->in_avail());
      cin.clear();
      temp->next = NULL;
      temp->prev = NULL;
      return temp;
}
//-------------------------------------------------------------
void outputList(Address **phead, Address **plast)      //Вывод списка на экран
{     Address *t = *phead;
      while (t)
      {     cout << t->name << ' ' << t->city << endl;
            t = t->next;
      }
      cout << "" << endl;
}
//-------------------------------------------------------------
void find(char name[NAME_SIZE], Address **phead)    // Поиск имени в списке
{     Address *t = *phead;
      while (t)
      {     if (!strcmp(name, t->name)) break;
            t = t->next;
      }
```

```
     if (!t)
          cerr << "Имя не найдено" << endl;
     else
          cout << t->name << ' ' << t->city << endl;
}
//-------------------------------------------------------------
void delet(char name[NAME_SIZE], Address **phead, Address **plast)  // Удаление имени
{    struct Address *t = *phead;
     while (t)
     {    if (!strcmp(name, t->name))  break;
          t = t->next;
     }
     if (!t)
          cerr << "Имя не найдено" << endl;
     else
     {    if (*phead == t)
          {    *phead = t->next;
               if (*phead)
                    (*phead)->prev = NULL;
               else
                    *plast = NULL;
          }
          else
          {    t->prev->next = t->next;
               if (t != *plast)
                    t->next->prev = t->prev;
               else
                    *plast = t->prev;
          }
          delete t;
```

```
            cout << "Элемент удален" << endl;
      }
}
//-------------------------------------------------------------
int main(void)
{     Address *head = NULL;
      Address *last = NULL;
      setlocale(LC_CTYPE, "Rus");
      while(true)
      {     switch (menu())
            {     case 1:  insert(setElement(), &head, &last);
                           break;
                  case 2:  {   char dname[NAME_SIZE];
                               cout << "Введите имя: ";
                               cin.getline(dname, NAME_SIZE-1,'\n');
                               cin.ignore(cin.rdbuf()->in_avail());
                               cin.sync();
                               delet(dname,&head, &last);
                           }
                           break;
                  case 3:  outputList(&head, &last);
                           break;
                  case 4:  {   char fname[NAME_SIZE];
                               cout << "Введите имя: ";
                               cin.getline(fname, NAME_SIZE - 1, '\n');
                               cin.ignore(cin.rdbuf()->in_avail());
                               cin.sync();
                               find(fname, &head);
                           }
                           break;
```

```
                case 5:  exit(0);
                default: exit(1);
            }
        }
        return 0;
}
```

2. В правой части приведены функции, осуществляющие запись информации в файл и считывание из файла.

Добавить эти функции в программу п.1 и внести соответствующие изменения в меню.

```
void writeToFile(Address **phead)       //Запись в файл
{     struct Address *t = *phead;
      FILE *fp;
      errno_t err = fopen_s(&fp, "mlist", "wb");
      if (err)
      {     cerr << "Файл не открывается" << endl;
            exit(1);
      }
      cout << "Сохранение в файл" << endl;
      while (t)
      {     fwrite(t, sizeof(struct Address), 1, fp);
            t = t->next;
      }
      fclose(fp);
}
//-------------------------------------------------------------
void readFromFile(Address **phead, Address **plast)          //Считывание из файла
{     struct Address *t;
      FILE *fp;
      errno_t err = fopen_s(&fp, "mlist", "rb");
      if (err)
      {     cerr << "Файл не открывается" << endl;
            exit(1);
      }
```

```
        while (*phead)
        {     *plast = (*phead)->next;
              delete *phead;
              *phead = *plast;
        }
        *phead = *plast = NULL;
        cout << "Загрузка из файла" << endl;
        while (!feof(fp))
        {     t = new Address();
              if (!t)
              {     cerr << "Ошибка выделения памяти" << endl;
                    return;
              }
              if (1 != fread(t, sizeof(struct Address), 1, fp)) break;
              insert(t, phead, plast);
        }
        fclose(fp);
}
```

2. Дополнить программу функцией в соответствии со своим вариантом из таблицы, представленной ниже.

№ вари-анта	Задание для выполнения
1	**deleteDouble()** – функция удаления повторяющихся (имеющих одинаковые поля или одно из полей) элементов
2	**deleteKFirst(int k)** – функция удаления **K** первых элементов списка.

№ варианта	Задание для выполнения
3	**deleteEveryM (int m)** – функция удаления каждого **M**-ого элемента списка.
4	**deleteKLast(int k)** – функция удаления **K** последних элементов списка.
5	**insertEnd(void* data)** – добавление нового элемента в конец списка
6	**findMin()** – функция поиска минимального элемента списка по одному из выбранных полей.
7	**findMax()** – функция поиска максимального элемента списка по одному из выбранных полей.
8	**oneByOne (Object &To, Object &from1, Object &from2)** - функцию, которая формирует список **To**, включив в него поочередно элементы из списков **from1** и **from2**.
9	**addXEnd (int x)** – функция добавление элемента **x** в конец списка.
10	**addXBegin (int x)** – функция добавления элемента **x** в начало списка.
11	**changeX (int i, x)** – функция замены i-го элемента списка элементом с заданным значением **x**.
12	**deleteX (int x)** – функция удаления первого встречающегося элемента с заданным значением **x**.
13	**countX (int x)** – функция подсчёта числа элементов списка с заданным значением **x**.
14	**returnN** – функция возвращения истинного значения, если список пуст и возвращение ложного в противном случае.
15	**addLEnd** – функция добавления в конец списка всех элементов некоторого списка **L**.
16	**addLBegin** – функция добавления в начало списка всех элементов некоторого списка **L**.

2. Дополнительное задание.

1. В приложении 2 приведен проект, в котором реализован проект с использованием *двусвязного списка*. На основе данного проекта разработать функции, которые предлагается создать в данном приложении.

2. Каждый элемент списка студентов содержит фамилию, имя, отчество, год рождения, курс, номер группы, оценки по пяти предметам. Упорядочить студентов по курсу, причем студенты одного курса должны располагаться в алфавитном порядке. Найти средний балл каждой группы по каждому предмету. Определить самого старшего студента и самого младшего студентов. Для каждой группы найти лучшего с точки зрения успеваемости студента.

3. N человек располагаются по кругу. Начав отсчет от первого, удаляют каждого k-го, смыкая круг после удаления. Определить порядок удаления людей из круга. Использовать линейный список.

Тест "Динамические структуры данных. Списки"

Контрольная работа 2

В начало практикума

Лабораторная работа № 10. Рекурсивные алгоритмы

Рекурсивная функция – это функция с такой организацией работы, при которой она вызывает сама себя. Рекурсия должна иметь внутри себя условие завершения. Рекурсивная функция (программа) может быть *линейной* (функция содержит единственный вызов самой себя), *смешанной* (две или более функций вызывают друг друга попеременно), *ветвящейся* (когда рекурсивный вызов содержится в теле функции более одного раза, либо производится в цикле), *вложенной* (имеется вызов функции внутри обращения к самой функции).

Задание	Краткие теоретические сведения
1. Изучить использование простых *рекурсивных* функций на примере программы, представленной в правой части. Определить признак конца ввода чисел, проанализировав текст программы. Добавить операторы, позволяющие определить, сколько раз функция **binPrn** вызывает сама себя.	``` #include <stdio.h> void binPrn(unsigned num) { if (num / 2) binPrn(num / 2); putchar(num % 2 + '0'); } int main(void) { int c; while (1) { printf("Number: "); if (scanf_s("%d", &c) != 1 || !c) break; binPrn(c); putchar('\n'); } return 0; } ``` Пример программы перевода десятичного целого числа в двоичный вид с использованием рекурсии.

2. Выполнить программу, разработанную с использованием рекурсии.

Реализовать алгоритм без использования рекурсии.

```
#include <iostream>
char s[100];
bool isPalindrom(char s[100]);

void main()
{
	setlocale(LC_CTYPE, "Russian");
	printf("\nВведите строку: ");
	gets_s(s);
	if (isPalindrom(s))
		printf("Строка - палиндром");
	else
		printf("Строка - не палиндром");
}
bool isPalindrom(char s[100])
{
	bool l;
	char s1[100];
	if (strlen(s) <= 1)
		return true;
	else
	{
		l = s[0] == s[strlen(s) - 1];
		strncpy_s(s1, s + 1, strlen(s) - 2);
		s1[strlen(s) - 2] = '\0';
		return l  &&  isPalindrom(s1);
	}
}
```

Пример. Определить, является ли заданная строка палиндромом, т.е. читается одинаково слева направо и справа налево.

Идея решения заключается в просмотре строки одновременно слева направо и справа налево и сравнении соответствующих символов. Если в какой-то момент символы не совпадают, делается вывод о том, что строка не является палиндромом, если же удается достичь середины строки и при этом все соответствующие символы совпали, то строка является палиндромом.

Строка также является палиндромом, если она пустая или состоит из одного символа.

3. Выполнить программу, приведенную в правой части. Записать ее условие.

Добавить любое слово в массив **w** и проанализировать результат работы программы.

Если некоторое слово в массиве **w** не подходит для решения задачи, то добавить операторы вывода сообщения о том, что его нужно изменить.

```
#include <iostream>
using namespace std;
char *w[] = {"ПАЛКА", "ЛЯГУШКА", "КАПЛЯ",
 "КАРТА", NULL};

bool test(char *s, char *r)
{
      int n;
      n = strlen(s);
      if (n == 0)
            return true;
      for (; *s != 0 && n > 1; s++, n--)
            if (strncmp(s, r, n) == 0)
                  return true;
      return false;
}

int step(char *lw)    // текущее слово
{
      setlocale(LC_CTYPE, "Russian");
      int n;
      for (n = 0; w[n] != NULL; n++)
            if (*w[n] != 0)
                  break;
      if (w[n] == NULL)
                  // цепочка выстроена
            return 1;
for (n = 0; w[n] != NULL; n++)
      {
            char *pw;
            if (*w[n] == 0)
                  continue;
            pw = w[n];
          // пустое слово – пропустить
            w[n] = "";
            if (test(lw, pw))
            {
                  if (step(pw))
                  //присоединено
                  {
                        cout << pw << "\n";
                        return 1;
                  }
            }
            w[n] = pw;
      }
      return 0;
}

void main()
{
      step("");
}
```

4. В соответствии со своим вариантом выполнить задания из таблицы, представленной ниже. В некоторых заданиях имеются ошибки: не выполняется условие завершения рекурсии. Изменить условие такой задачи с тем, чтобы рекурсия выполнялась.

№	Условие задачи	Пояснения к алгоритму
1	Разработать программу, реализующую рекурсивную функцию подсчета количества **x(m)** разбиений натурального числа **m** в виде суммы натуральных чисел.	Для **m = 4** разбиения имеют вид: 4, 3+1, 2+2, 2+1+1, 1+1+1+1. Таким образом, **x(m) = 5**. Функция подсчета **P(m, n)** количества разбиений натурального числа **m** со слагаемыми, не превосходящими **n**, определяется следующим образом: $$P(m,n)=\begin{cases}1, & n=1 \text{ или } m=1\\ P(m,m), & n>m\\ P(m,m-1)+1, & m=n\\ P(m,n-1)+P(m-n,n), & n<m\end{cases}$$ При этом **x(m) = P(m, m).**
2	Разработать программу, реализующую рекурсивный алгоритм вычисления значений **F(m, n**) для любых целых не отрицательных аргументов **m** и **n**.	Функция имеет следующий вид: $$F(m,n)=\begin{cases}\max\{n,m\}, & \text{если } (n+m) \text{ четная},\\ F\left(\frac{n+m+1}{2}, n+1\right)+F\left((m,\frac{n+m+1}{2})\right), & \text{в остальных случаях}\end{cases}$$

№	Условие задачи	Пояснения к алгоритму
3	Вычислить выражение в правой части, используя рекурсию.	$\sqrt{1+(n+1)\sqrt{1+(n+2)\sqrt{1+(n+3)\sqrt{1+.....(n+n)}}}}$
4	Задан прямоугольник со сторонами **a** и **b** (**a, b** − натуральные числа). Разбить его на части с помощью квадратов и определить, сколько квадратов получится, если каждый раз выбирается самый большой квадрат.	Пусть, например **a > b**. Тогда, отрезав от прямоугольника **floor(a/b)** квадратов со сторонами длиной **b**, снова окажемся перед исходной задачей, в которой **a = b** и **b = mod(a, b)** (**b = a - floor(a / b) · b**). В качестве базы рекурсии можно взять случай **b = 0**. Если **numsq(a, b)** − функция, возвращающая решение задачи при заданных длинах **a** и **b** сторон исходного прямоугольника, то $numcq(a,b)=\begin{cases}0, b=0\\ floor(a/b)+numcq(b,mod(a,b)),\ b\neq 0\end{cases}$
5	Бином Ньютона определяется как: $C_n^m = n! / (m!(n-m)!)$ Рекурсивно описать функцию **C(m, n)**, где **0 <= m <= n** для биноминального коэффициента **Cn**.	Формулы имеют следующий вид: $C_n^0 = C_n^n = 1;$ $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$
6	Разработать программу, реализующую рекурсивный алгоритм вычисления **A(m, n)** для любых целых не отрицательных **m** и **n**.	$A(m,n)=\begin{cases}A(0,n)=n+1\\ A(m,0)+A(m-1,1)\ \text{если}\ m>0\\ A(m,n)+A(m-1,A(m,n-1)),\text{если}\ m,n>0\end{cases}$

№	Условие задачи	Пояснения к алгоритму
7	Вычислить выражение в правой части, используя рекурсию.	$y = \sin x + \sin 2x + \sin 3x + \ldots + \sin nx$
8	Пусть функция **f(x)** вещественной переменной **x** непрерывна на отрезке **[a, b]** и **f(a)×f(b) < 0**. Разработать рекурсивную программу нахождения на отрезке **[a, b]** какого-либо вещественного корня.	Для определения корня можно использовать метод деления отрезка пополам. С использованием рекурсии: $dicho(f,a,b,e)=\begin{cases} c\ (c=(a+b)/2), (b-a)\le 2e \\ dicho(f,a,c,e),\ f(a)f(c)\le 0 \\ dicho(f,c,b,e),\ f(a)f(c)>0 \end{cases}$ Здесь **e** – заданная точность решения.
9	Пусть для целых неотрицательных чисел **n**, **m** разрешены операции нахождения последующего числа **(n + 1)** и предыдущего числа **n - 1 (n > 0)**. С помощью рекурсивных функций определить операции нахождения суммы **(n + m)**, разности **(n – m)**, умножения **(n × m)**, возведения в степень **n ^ m (n > 0)**.	Для суммы **a + b** очевидно соотношение: $a+b=\begin{cases} a,\ b=0 \\ (a+1)+(b\text{-}1),\ b\neq 0 \end{cases}$ Оно задает одновременно и базу рекурсии и правило декомпозиции. По нему построена функция, определяющая сложение: $plus(a,b)=\begin{cases} a,\ b=0 \\ plus(a+1, b\text{-}1),\ b\neq 0 \end{cases}$ Разность: $minus(a,b)=\begin{cases} a,\ b=0 \\ min\,us(a\text{-}1, b\text{-}1),\ b\neq 0 \end{cases}$

№	Условие задачи	Пояснения к алгоритму
		Умножение: $\text{multi}(a,b)=\begin{cases}0, & b=0\\ a, & b=1\\ \text{plus}(\text{multi}(a,b\text{-}1),a), & b\neq 0 \text{ или } b\neq 1\end{cases}$ Возведение в степень: $\text{power}(a,b)=\begin{cases}1, & b=0\\ \text{multi}(a,\text{power}(a,b\text{-}1)), & b\neq 0\end{cases}$
10	Разработать программу, реализующую рекурсивный алгоритм вычисления значений **F(m, n)** для любых целых не отрицательных аргументов **m** и **n**.	Функция имеет следующий вид: $F(m,n)=\begin{cases}n+1, \text{если } m=0 \text{ или } n=0,\\ F(m-1,F(m,n-1)), \text{в остальных случаях}\end{cases}$
11	Разработать программу, реализующую рекурсивный алгоритм для вычисления значений полиномов $\mathbf{S_n(x)}$.	Функция имеет следующий вид: $S_n(x)=\begin{cases}0, \textit{если } n=0,\\ 2x, \textit{если } n=1,\\ \frac{2n}{n-1}S_{n-1}(x)+\frac{n-1}{2n}S_{n-2}(x), \textit{если } n>1\end{cases}$

№	Условие задачи	Пояснения к алгоритму
12	Вычислить выражение в правой части, используя рекурсию.	$y = \cos x + \cos 2x + \cos 3x + \ldots + \cos nx$
13	Разработать программу, реализующую рекурсивную функцию подсчета количества всех положительных делителей натурального числа **n**.	Рассмотрим более общую задачу подсчета для натурального числа **n** количества всех его положительных делителей, меньших или равных заданному натуральному числу **x**. Пусть **dn(n)** и **dnx(n, x)** − функции для решения исходной и обобщенной задач. Рекурсивную функцию **dnx(n, x)**, по которой последовательно подвергаются испытанию на делители **n** все числа от 1 до **x** включительно, можно определить так: $$dnx(n,x)=\begin{cases}1, x=1\\ dnx(n, x\text{-}1)+\begin{cases}1, если (n\ \%\ x)=0 \text{ и } x>1\\ 0, если (n\ \%\ x)\neq 0 \text{ и } x>1\end{cases}\end{cases}$$ Очевидно, что **dn(n) = dnx(n, n)**.
14	Разработать программу, реализующую рекурсивную функцию **f(x, n)**, вычисляющую величину $x^n/n!$ при любом вещественном **x** и любом неотрицательном целом **n**.	Для вычисления $x^{n-1}/(n-1)!$ надо рекурсивно обратиться к **f(x, n-1),** а затем полученную величину умножить на **x/n**, чтобы получить значение **f(x, n)**. Функция **f(x, n)** принимает следующий вид:

№	Условие задачи	Пояснения к алгоритму
		$f(x,n)=\begin{cases}1, n=0\\ x, n=1\\ x*x / n /(n-1)f(x,n-2), & n\neq 0 \text{ или } n\neq 1\end{cases}$
15	Разработать программу, реализующую рекурсивный алгоритм вычисления значений **S(x)** для любых целых не отрицательных значений **x**.	$S(x)=\begin{cases}x+10, \text{если } x>100\\ S(S(x+4)), \text{в остальных случаях}\end{cases}$
16	Разработать программу, реализующую рекурсивный алгоритм вычисления значений **F(n)** для любых целых не отрицательных аргументов **n**.	$F(n)=\begin{cases}1, \text{если } n=0\\ -1, \text{если } n<m\\ 2*F(n-1,m), \text{в остальных случаях}\end{cases}$

5. К номеру своего варианта прибавить число 2 и написать программу для новых исходных данных (для вариантов 15, 16 перейти к вариантам 1, 2).

6. Дополнительные задания.

1. Ввести цифру **A**, записать в файл все возможные числа, состоящие из цифр, не превышающих или равных **A**. Количество цифр в числах должно быть равно **A**.

Примечание: использовать дополнительный массив.

2. Задача проведения границы на карте («создание военных блоков»). Страны на карте заданы матрицей смежности. Если страны **i**, **j** имеют на карте общую границу, то элемент матрицы **A[i, j]** равен 1, иначе 0.

Необходимо разбить страны на две группы так, чтобы количество пар смежных стран из противоположных групп было минимальным.

3. Дано **n** различных натуральных чисел (**n = 5**). Напечатать все перестановки этих чисел.

4. По заданному числу **n** определить символ, который стоит на **n**-ом месте последовательности, получившейся после шага с номером 26.

Тест "Рекурсивные алгоритмы"

В начало практикума

Лабораторная работа № 11. Бинарные деревья

Дерево – это структура, имеющая следующие свойства:
– существует единственный элемент (узел, вершина), на который не ссылается никакой другой и который называется *корнем*;
– начиная с корня и следуя по определенной цепочке указателей, можно осуществить доступ к любому элементу структуры;
– на каждый элемент, кроме корня, имеется единственная ссылка.

Бинарное дерево поиска – это упорядоченное дерево, каждая вершина которого имеет **не более двух** поддеревьев: в левом поддереве содержатся ключи, имеющие значения, **меньшие**, чем значение данного узла, в правом поддереве содержатся ключи, имеющие значения, **большие**, чем значение данного узла.

Задание	Краткие теоретические сведения
1. Изучить работу с *бинарным деревом*, выполнив программу, записанную в правой части. В программе осуществляется добавление элемента в дерево и вывод дерева на экран с поворотом на 90 градусов влево.	

```
#include <iostream>
#include<conio.h>
using namespace std;
struct Node
{    int data;              //Информационное поле
     Node *left, *right;    //Указатели на левую и правую ветви дерева
};
Node* tree = nullptr;

void insert(int a, Node **t)  //Добавление элемента a
{    if ((*t)==NULL)               //если дерева нет, то создается элемент
     {    (*t) = new Node;
          (*t)->data = a;
          (*t)->left = (*t)->right = NULL;
          return;
     }
     if (a > (*t)->data)  //дерево есть, если a больше текущего
```

```
            insert(a, &(*t)->right);  //то элемент помещается вправо
      else
            insert(a, &(*t)->left);   //иначе - влево
}

void print(Node *t, int u)    //Вывод на экран
{     if (t == NULL)  return;
      else
      {     print(t->left, ++u);     //левое поддерево
            for (int i = 0; i < u; ++i)
                  cout << "|";
            cout << t->data << endl;
            u--;
      }
      print(t->right, ++u);          // правое поддерево
}

void main()
{     setlocale(LC_CTYPE, "Russian");
      int count, temp;
      cout << "Введите количество элементов  "; cin >> count;
      for (int i = 0; i<count; ++i)
      {     cout << "Введите число  "; cin >> temp;
            insert(temp, &tree);
      }
      cout << "ваше дерево\n";
      print(tree, 0);
      _getch();
}
```

2. В правой части приведена программа, которая осуществляет построение бинарного дерева, каждый элемент которого состоит из ключа (целое число) и слова размером не более 4 символов. В программе осуществляется добавление элемента, удаление элемента, поиск элемента по ключу, вывод дерева на экран, очистка дерева.

При вводе информации признак окончания – ввод отрицательного ключа.

Выполнить программу и написать комментарии к операторам.

Пример. Определить количество записей в бинарном дереве, начинающихся с введенной с клавиатуры буквы.

```
#include <iostream>
using namespace std;
struct Tree          //дерево
{   int key;         //ключ
    char text[5];    //текст - не более 4 букв
    Tree *Left, *Right;
};

Tree* makeTree(Tree *Root);       //Создание дерева
Tree* list(int i, char *s);       //Создание нового элемента
Tree* insertElem(Tree *Root, int key, char *s);  //Добавление нового элемента
Tree* search(Tree* n, int key);   //Поиск элемента по ключу
Tree* delet(Tree *Root, int key); //Удаление элемента по ключу
void view(Tree *t, int level);    //Вывод дерева
int count(Tree *t, char letter);  //Подсчет количества слов
void delAll(Tree *t);             //Очистка дерева

int c = 0;         //количество слов
Tree *Root = NULL; //указатель корня

void main()
{  setlocale(0, "Russian");
   int key, choice, n;
   Tree *rc; char s[5], letter;
   for (;;)
    {   cout<<"1 - создание дерева\n";
        cout<<"2 - добавление элемента\n";
```

```
cout<<"3 - поиск по ключу\n";
cout<<"4 - удаление элемента\n";
cout<<"5 - вывод дерева\n";
cout<<"6 - подсчет количества букв\n";
cout<<"7 - очистка дерева\n";
cout<<"8 - выход\n";
cout<<"ваш выбор?\n";
cin>>choice;
cout<<"\n";
switch (choice)
{     case 1:  Root = makeTree(Root); break;
      case 2:  cout<<"\nВведите ключ: "; cin>>key;
               cout<<"Введите слово: "; cin>>s;
               insertElem(Root, key, s); break;
      case 3:  cout<<"\nВведите ключ: ";  cin>>key;
               rc = search(Root, key);
               cout<< "Найденное слово= ";
               puts(rc->text); break;
      case 4:  cout<<"\nВведите удаляемый ключ: "; cin>>key;
               Root = delet(Root, key);  break;
      case 5:  if(Root->key >= 0)
               {  cout<<"Дерево повернуто на 90 град. влево"<<endl;
                  view(Root, 0);
               }
               else cout<<"Дерево пустое\n"; break;
      case 6:  cout<<"\nВведите букву: "; cin>>letter;
               n = count(Root, letter);
               cout<<"Количество слов, начинающихся с буквы "<<letter;
               cout <<" равно "<<n<<endl; break;
      case 7:  delAll(Root); break;
```

```
                case 8:  exit(0);
            }
    }
}

Tree* makeTree(Tree *Root)    //Создание дерева
{  int key; char s[5];
   cout<<"Конец ввода - отрицательное число\n\n";
   if ( Root == NULL )  // если дерево не создано
   {   cout<<"Введите ключ корня: "; cin>>key;
       cout<<"Введите слово корня: "; cin>>s;
       Root = list(key, s);  // установка указателя на корень
   }
   while(1)                //добавление элементов
   {    cout<<"\nВведите ключ: ";  cin>>key;
        if  (key < 0) break;       //признак выхода (ключ < 0)
        cout<<"Введите слово: ";  cin>>s;
        insertElem(Root, key, s);
   }
   return Root;
}

Tree* list(int i, char *s)     //Создание нового элемента
{  Tree *t =  new Tree[sizeof(Tree)];
   t -> key = i;
   for(i = 0;  i < 5;  i++)
       *((t -> text)+i) = *(s+i);
   t -> Left = t -> Right = NULL;
   return t;
}
```

```
Tree* insertElem(Tree *t, int key, char *s)  //Добавление нового элемента
{   Tree *Prev;          // Prev - элемент перед текущим
    int find = 0;         // признак поиска
    while ( t && ! find)
    {   Prev = t;
        if( key == t->key)
              find = 1;     //ключи должны быть уникальны
        else
              if ( key < t -> key ) t = t -> Left;
              else t = t -> Right;
    }
    if (! find)               //найдено место с адресом Prev
    {   t = list(key, s);           //создается новый узел
        if ( key < Prev -> key )  // и присоединяется либо
              Prev -> Left = t;    //переход на левую ветвь,
        else
              Prev -> Right = t;   // либо на правую
   }
   return t;
}

Tree* delet(Tree *Root, int key)  //Удаление элемента по ключу
{  // Del, Prev_Del - удаляемый элемент и его предыдущий ;
   // R, Prev_R - элемент, на который заменяется удаленный, и его родитель;
   Tree *Del, *Prev_Del, *R, *Prev_R;
   Del = Root;
   Prev_Del = NULL;
   while (Del != NULL && Del -> key != key)//поиск элемента и его родителя
   {   Prev_Del = Del;
```

```
            if (Del->key > key)
                  Del = Del->Left;
            else
                  Del = Del->Right;
      }
      if (Del == NULL)              // элемент не найден
      {   puts("\nНет такого ключа");
          return Root;
      }
      if (Del -> Right == NULL) // поиск элемента R для замены
            R = Del->Left;
      else
            if (Del -> Left == NULL)
                  R = Del->Right;
            else
            {     Prev_R = Del; //поиск самого правого элемента в левом поддереве
                  R = Del->Left;
                  while (R->Right != NULL)
                  {     Prev_R = R;
                        R = R->Right;
                  }
                  if( Prev_R == Del) // найден элемент для замены R и его родителя Prev_R
                      R->Right = Del->Right;
                  else
                  {     R->Right = Del->Right;
                        Prev_R->Right = R->Left;
                        R->Left = Prev_R;
                  }
            }
      if (Del== Root) Root = R;  //удаление корня и замена его на R
```

```
    else
    // поддерево R присоединяется к родителю удаляемого узла
         if (Del->key < Prev_Del->key)
              Prev_Del->Left = R; // на левую ветвь
         else
              Prev_Del->Right = R;  // на правую ветвь
    int tmp = Del->key;
    cout<<"\nУдален элемент с ключом "<<tmp <<endl;
    delete Del;
    return Root;
}

Tree* search(Tree* n, int key)  //Поиск элемента по ключу
{  Tree* rc = n;
    if (rc != NULL)
    {    if (key < (key, n->key))
             rc = search(n->Left, key);
         else
            if (key > (key, n->key))
                rc = search(n->Right, key);
     }
     else
         cout<<"Нет такого элемента\n";
     return rc;
}

int count(Tree *t, char letter) //Подсчет количества слов
{  if ( t )
    {    count ( t -> Right, letter);
         if(*(t->text) == letter)
```

```
            c++;
        count( t -> Left, letter);
    }
    return c;
}

void view ( Tree *t, int level ) //Вывод дерева
{  if (t)
   {    view (t -> Right, level+1);     //вывод правого поддерева
        for (int i = 0; i < level; i++)
            cout<<"   ";
        int tm = t->key;
        cout<<tm<<' ';
        puts(t->text);
         view(t -> Left, level+1);  //вывод левого поддерева
   }
}

void delAll(Tree *t) //Очистка дерева
{   if (t!= NULL)
    {  delAll(t -> Left);
       delAll(t -> Right);
      delete t;
    }
}
```

3. Разработать программу работы с бинарным деревом, в которую включить основные функции манипуляции данными и функцию в соответствии со своим вариантом из таблицы, представленной ниже.

№ варианта	Задание для выполнения
1	Вершина бинарного дерева содержит ключ, три целых числа и два указателя на потомков. Написать функцию удаления вершины с минимальной суммой трех целых значений узла.
2	Дан указатель **p1** на корень бинарного дерева. Написать функцию вывода количества листьев дерева, которые являются правыми дочерними вершинами.
3	Вершина бинарного дерева содержит ключ, строку и два указателя на потомков. Написать функцию вывода всех элементов дерева по уровням: корень дерева, вершины 1-го уровня, вершины 2-го уровня, ...
4	Вершина бинарного дерева содержит ключ, строку и два указателя на потомков. Написать функцию, которая подсчитывает число ветвей от корня до ближайшей вершины с заданным ключом, и выводит их.
5	Дан указатель **p1** на корень непустого дерева. Написать функцию вывода количества вершин дерева, являющихся левыми дочерними вершинами (корень дерева не учитывать).
6	В текстовом файле построчно записаны целые числа. Написать функцию, которая по файлу F, все элементы которого различны, строит соответствующее бинарное дерево поиска T, и вторую функцию, которая записывает в файл G элементы дерева поиска T в порядке их возрастания.
7	Дан указатель **p1** на корень непустого дерева. Написать функцию вывода количества листьев данного дерева.
8	Вершина бинарного дерева содержит ключ, 2 целых числа и два указателя на потомков. Написать функцию удаления вершины с максимальной суммой 2 целых значений узла.
9	Дан указатель **p1** на корень непустого дерева и число **k**. Написать функцию вывода количества вершин дерева, значение которых равно **k**.

№ варианта	Задание для выполнения
10	Дан указатель **p1** на корень непустого дерева. Написать функцию вывода вывода суммы значений всех вершин данного дерева.
11	Вершина бинарного дерева содержит ключ, строку и два указателя на потомков. Написать функцию определения числа ветвей **n**-го уровня этого дерева и вывода этих элементов на экран.
12	Дан указатель **p1** на корень непустого дерева. Написать функцию определения количества узлов с четными ключами.
13	Дан указатель **p1** на корень непустого дерева. Написать функцию, в которой для каждого из уровней данного дерева, начиная с нулевого, вывести сумму значений вершин, находящихся на этом уровне. Считать, что глубина дерева не превосходит 10.
14	Вершина бинарного дерева содержит ключ и два указателя на потомков. Написать функцию вычисления среднего арифметического всех элементов дерева.
15	Вершина бинарного дерева содержит ключ, строку и два указателя на потомков. Написать функцию определения количества узлов правой ветви дерева.
16	Дан указатель **p1** на корень непустого дерева. Написать функцию вывода суммы значений всех листьев данного дерева.

В начало практикума

Лабораторная работа № 12. Разработка проекта с использованием бинарного дерева

Проект с использованием бинарного дерева в данной работе состоит из трех частей. Предполагается, что в дереве не более 10 уровней. Дублирование ключей не допускается.

Задание	Краткие теоретические сведения
1. В правой части записана *главная функция* проекта и функции, выполняющие некоторые действия с данными. Дополнить комментарии к программе.	

```
#include "stdafx.h"
#include "Tree.h"
#include <fstream>
using namespace std;
struct NodeTree
{    int  key;
};
//-------------------------------
btree::CMP cmpfnc(void* x, void* y)    // Сравнение
{    btree::CMP rc = btree::EQUAL;
     if (((NodeTree*)x)->key  <  ((NodeTree*)y)->key)
          rc = btree::LESS;
     else
          if (((NodeTree*)x)->key  >((NodeTree*)y)->key)
               rc = btree::GREAT;
     return rc;
}
//-------------------------------
void printNode(void* x)               // Вывод при обходе
{    cout << ((NodeTree*)x)->key << ends;  }
//-------------------------------
bool buildTree(char *FileName, btree::Object& tree) //Построение дерева из файла
{    bool rc = true;
```

```
        FILE *inFile;
        fopen_s(&inFile, FileName, "r");
        if (inFile == NULL)
        {    cout << "Ошибка открытия входного файла" << endl;
             rc = false; return rc;
        }
        while (!feof(inFile)) // заполнение дерева
        {    int num;
             fscanf_s(inFile, "%d", &num, 1);
             NodeTree *a = new NodeTree();
             a->key = num;
             tree.insert(a);
        }
        fclose(inFile);
         return rc;
    }
    FILE * outFile;
    //-------------------------------
    void saveToFile(void *x)              // Запись одного элемента в файл
    {    NodeTree *a = (NodeTree*)x;
         int q = a->key;
         fprintf(outFile, "%d\n", q);
    }
    //-------------------------------
    void saveTree(btree::Object &tree, char *FileName)    //Сохранение дерева в файл
    {    fopen_s(&outFile, FileName, "w");
         if (outFile == NULL)
         {     cout << "Ошибка открытия выходного файла" << endl;
               return;
         }
```

```
        tree.Root->scanDown(saveToFile);
        fclose(outFile);
    }
    //-------------------------------
    int _tmain(int argc, _TCHAR* argv[])
    {   setlocale(LC_CTYPE, "Russian");
        btree::Object demoTree = btree::create(cmpfnc);
        int k, choice;
        NodeTree a1 = { 1 }, a2 = { 2 }, a3 = { 3 }, a4 = { 4 }, a5 = { 5 }, a6 = { 6 };
        bool rc = demoTree.insert(&a4);   //           4
        rc = demoTree.insert(&a1);        //   1
        rc = demoTree.insert(&a6);        //                   6
        rc = demoTree.insert(&a2);        //     2
        rc = demoTree.insert(&a3);        //        3
        rc = demoTree.insert(&a5);        //                 5
        for (;;)
        {     NodeTree *a = new NodeTree;
              cout << "1 - вывод дерева на экран" << endl;
              cout << "2 - добавление элемента" << endl;
              cout << "3 - удаление элемента" << endl;
              cout << "4 - сохранить в файл" << endl;
              cout << "5 - загрузить из файла" << endl;
              cout << "6 - очистить дерево" << endl;
              cout << "0 - выход" << endl;
              cout << "сделайте выбор" << endl; cin >> choice;
              switch (choice)
              {    case 0:    exit(0);
                   case 1:    if (demoTree.Root)
                                   demoTree.Root->scanByLevel(printNode);
                              else
```

```
                                        cout << "Дерево пустое" << endl;
                                    break;
                            case 2:    cout << "введите ключ" << endl;  cin >> k;
                                       a->key = k;
                                       demoTree.insert(a);
                                       break;
                            case 3:    cout << "введите ключ" << endl;  cin >> k;
                                       a->key = k;
                                       demoTree.deleteByData(a);
                                       break;
                            case 4:    saveTree(demoTree, "G.txt");
                                       break;
                            case 5:    buildTree("G.txt", demoTree);
                                       break;
                            case 6:    while (demoTree.Root)
                                            demoTree.deleteByNode(demoTree.Root);
                                       break;
                        }
                    }
                    return 0;
                }
```

2. В правой части записан *программный модуль* **Tree.cpp**.

Написать комментарии к программному коду.

Программный модуль **Tree.cpp**

```
#include "stdafx.h"
#include "Tree.h"
namespace btree      // бинарное дерево, не допускается дублирование ключей
{    Object create(CMP(*f)(void*, void*))
     {     return *(new Object(f));   }
     //------------------------------
     Node* Node::min()
```

```
{     Node* rc = this;
      if (rc->left != NULL)
            rc = rc->left->min();
      return rc;
}
//------------------------------
Node* Node::next()
{     Node* rc = this, *x = this;
      if (rc->right != NULL)
            rc = rc->right->min();
      else
      {     rc = this->parent;
            while (rc != NULL && x == rc->right)
            {     x = rc;
                  rc = rc->parent;
            }
      }  return rc;
}
//------------------------------
void Node::scanDown(void(*f)(void* n))
{     f(this->data);
      std::cout << std::endl;
      if (this->left != NULL)
            this->left->scanDown(f);
      if (this->right != NULL)
            this->right->scanDown(f);
}
//------------------------------
Node* Object::search(void* d, Node* n)
{     Node* rc = n;
```

```
        if (rc != NULL)
        {    if (isLess(d, n->data))
                  rc = search(d, n->left);
             else
                  if (isGreat(d, n->data))
                       rc = search(d, n->right);
        } return rc;
   }
   //-------------------------------
   bool Object::insert(void* d)
   {    Node* x = this->Root, *n = NULL;
        bool rc = true;
        while (rc == true && x != NULL)
        {    n = x;
             if (isLess(d, x->data))
                  x = x->left;          //выбор куда идти - влево или вправо
             else
                  if (isGreat(d, x->data))
                       x = x->right;
                  else
                       rc = false;
        }
        if (rc == true && n == NULL)
             this->Root = new  Node(NULL, NULL, NULL, d);
        else
             if (rc == true && isLess(d, n->data))
                  n->left = new  Node(n, NULL, NULL, d);
             else
                  if (rc == true && isGreat(d, n->data))
                       n->right = new  Node(n, NULL, NULL, d);
```

```
        return rc;
    };
    //--------------------------------
    bool  Object::deleteByNode(Node* n)
    {    bool rc = true;
         if (rc = (n != NULL))
         {    if (n->left == NULL  &&  n->right == NULL)       //если потомков нет
              {      if (n->parent == NULL)
                          this->Root = NULL;   //обнуление корня
                     else
                          if (n->parent->left == n)
                               n->parent->left = NULL;
                          else
                              n->parent->right = NULL;
                     delete n;
              }
              else
                   if (n->left == NULL  &&  n->right != NULL) //только правый потомок
                        {    if (n->parent == NULL)
                                  this->Root = n->right;
                             else
                                  if (n->parent->left == n)
                                       n->parent->left = n->right;
                                  else
                                       n->parent->right = n->right;
                             n->right->parent = n->parent;
                             delete n;
                        }
                    else
                        if (n->left != NULL &&n->right == NULL) //только левый потомок
```

```
                              {     if (n->parent == NULL)
                                          this->Root = n->left;
                                    else
                                          if (n->parent->right == n)
                                                n->parent->left = n->left;
                                          else
                                                n->parent->right = n->left;
                                    n->left->parent = n->parent;
                                    delete n;
                              }
                              else                                        //если есть оба потомка
                                    if (n->left != NULL  &&  n->right != NULL)
                                    {     Node* x = n->next();
                                          n->data = x->data;
                                          rc = deleteByNode(x);
                                    }
          } return  rc;
     }
     //------------------------------
     void Node::scanLevel(void(*f)(void* n), int i)  //Вывести вершины уровня
     {     if (this->left != NULL)
                this->left->scanLevel(f, i);
          if (this->getLevel() == i)
                f(this->data);
          if (this->right != NULL)
                this->right->scanLevel(f, i);
     }
     //------------------------------
     int Node::getLevel()
     {     Node *rc = this;
```

```
            int q = 0;
            while (rc->parent != NULL)
            {     rc = rc->parent;
                  q++;
            }
            return q;
      }
      //-------------------------------
      void Node::scanByLevel(void(*f)(void* n))
      {     for (int i = 0; i < 10; i++)
            {     std::cout << '\t';
                  this->scanLevel(f, i);
                  std::cout << '\n';
            }
      }
}
```

3. В правой части записан *заголовочный* файл **Tree.h**.

Написать комментарии к программному коду.

Заголовочный файл **Tree.h**

```
#pragma once
#include <iostream>
namespace btree
{     enum CMP
      {     LESS = -1, EQUAL = 0, GREAT = 1 };
      struct Node
      {     Node* parent;            // указатель на родителя
            Node* left;              // указатель на левую ветвь
            Node* right;             // указатель на правую ветвь
            void* data;              //  данные
            Node(Node* p, Node* l, Node* r, void* d) // конструктор
```

```
        {     parent = p;
              left = l;
              right = r;
              data = d;
        }
        Node* next();               // следующий по ключу
        Node* prev();               // предыдущий по ключу
        Node* min();                // минимум в поддереве
        Node* max();                // максимум в поддереве
        void scanDown(void(*f)(void* n));    // обход поддерева сверху вниз
        void scan(int(*f)(void* n));
        void scanLevel(void(*f)(void* n), int);
        int getLevel();
        void scanByLevel(void(*f)(void* n));
};
struct Object         // Интерфейс бинарного дерева
{     Node* Root;                   // указатель на корень
      CMP(*compare)(void*, void*);  // функция сравнения
      Object(CMP(*f)(void*, void*))
      {     Root = NULL;
            compare = f;
      };
      bool isLess(void* x1, void* x2) const
      {     return compare(x1, x2) == LESS; };
      bool isGreat(void* x1, void* x2)const
      {     return compare(x1, x2) == GREAT; };
      bool isEqual(void* x1, void* x2)const
      {     return compare(x1, x2) == EQUAL; };
      bool insert(void* data);          // добавить элемент
      Node* search(void* d, Node* n);   // найти элемент
```

```
                Node* search(void* d)
                {     return search(d, Root);  };
                bool deleteByNode(Node* e);          // удалить по адресу элемента
                bool deleteByData(void* data)        // удалить по ключу
                {     return deleteByNode(search(data)); };
            };
            Object create(CMP(*f)(void*, void*));      // Создать бинарное дерево
    };
```

4. Добавить к проекту функции смешанного и нисходящего обхода дерева с выводом на консоль, проверки сбалансированности дерева и функцию в соответствии с вариантом из таблицы, представленной в лабораторной работе № 11, изменив ее так, чтобы функция соответствовала проекту данной лабораторной работы.

5. Дополнительные задания.

1. Оператор мобильной связи организовал базу данных абонентов, содержащую сведения о телефонах, их владельцах и используемых тарифах, в виде бинарного дерева. Разработать программу, которая обеспечивает начальное формирование базы данных в виде бинарного дерева; производит вывод всей базы данных; поиск владельца по номеру телефона; выводит наиболее востребованный тариф (по наибольшему числу абонентов).

2. Дано N чисел, N > 0. Создать дерево из N вершин, в котором каждая левая дочерняя вершина является листом, а правая дочерняя вершина является внутренней. Для каждой внутренней вершины вначале создавать левую дочернюю вершину, а затем правую (если она существует); каждой создаваемой вершине присваивать очередное значение из исходного набора.

3. Изучить работу с красно-черными деревьями. Реализовать основные операции над красно-черным деревом. В красно-черном дереве найти путь от корня к некоторому листу, содержащий минимальное количество красных вершин.

Тест "Бинарные деревья"

В начало практикума

Лабораторная работа № 13. Бинарные кучи

Бинарная куча (binary heap) представляет собой бинарное дерево, для которого выполняется основное свойство кучи: приоритет каждой вершины **больше** приоритетов её потомков. В простейшем случае приоритет можно считать равным значению.

Реализация бинарной кучи возможна на основе *массива* и на основе *списка*.

Задание	Краткие теоретические сведения
1. Пункты 1, 2, 3 содержат программный код проекта, в котором реализована бинарная куча, которая представлена в виде *массива*. В правой части данного пункта записан *заголовочный файл* **Heap.h**. Написать комментарии к программному коду.	`#pragma once` `struct AAA` `{  int x;` `  void print();` `  int getPriority() const;` `};` `namespace heap` `{  enum CMP` `  {  LESS = -1, EQUAL = 0, GREAT = 1 };` `  struct Heap` `  {  int size;` `  int maxSize;` `  void** storage;  // данные` `  CMP(*compare)(void*, void*);` `  Heap(int maxsize, CMP(*f)(void*, void*))` `  {  size = 0;` `  storage = new void *[maxSize = maxsize];` `  compare = f;` `  };` `  int left(int ix);`

```
        int right(int ix);
        int parent(int ix);
        bool isFull() const
        { return (size >= maxSize); };
        bool isEmpty() const
        { return (size <= 0); };
        bool isLess(void* x1, void * x2) const
        { return compare(x1, x2) == LESS; };
        bool isGreat(void* x1, void* x2) const
        { return compare(x1, x2) == GREAT; };
        bool isEqual(void* x1, void* x2) const
        { return compare(x1, x2) == EQUAL; };
        void swap(int i, int j);
        void heapify(int ix);
        void insert(void* x);
        void* extractMax();
        void scan(int i) const;
    };
    Heap create(int maxsize, CMP(*f)(void*, void*));
};
```

2. В правой части записан *программный модуль* **Heap.cpp**.

Написать комментарии к программному коду.

```
#include "stdafx.h"
#include "Heap.h"
#include <iostream>
#include <iomanip>
void AAA::print()
{    std::cout << x; }
int AAA::getPriority() const
{    return x;  }
```

```
namespace heap
{    Heap create(int maxsize, CMP(*f)(void*, void*))
     {    return *(new Heap(maxsize, f)); }
     int Heap::left(int ix)
     {    return (2 * ix + 1 >= size) ? -1 : (2 * ix + 1); }
     int Heap::right(int ix)
     {    return (2 * ix + 2 >= size) ? -1 : (2 * ix + 2); }
     int Heap::parent(int ix)
     {    return (ix + 1) / 2 - 1;  }
     void Heap::swap(int i, int j)
     {    void* buf = storage[i];
          storage[i] = storage[j];
          storage[j] = buf;
     }
     void Heap::heapify(int ix)
     {    int l = left(ix), r = right(ix), irl = ix;
          if (l > 0)
          {    if (isGreat(storage[l], storage[ix])) irl = l;
               if (r > 0 && isGreat(storage[r], storage[irl])) irl = r;
               if (irl != ix)
               {    swap(ix, irl);
                    heapify(irl);
               }
          }
     }
     void Heap::insert(void* x)
     {    int i;
          if (!isFull())
          {    storage[i = ++size - 1] = x;
               while (i > 0 && isLess(storage[parent(i)], storage[i]))
```

```
            {     swap(parent(i), i);
                  i = parent(i);
            }
      }
}
void* Heap::extractMax()
{     void* rc=nullptr;
      if (!isEmpty())
      {     rc = storage[0];
            storage[0] = storage[size - 1];
            size--;
            heapify(0);
      } return rc;
}
void Heap::scan(int i) const     //Вывод значений элементов на экран
{     int probel = 20;
      std::cout << '\n';
      if (size == 0)
            std::cout << "Куча пуста";
      for (int u = 0, y = 0; u < size; u++)
      {     std::cout << std::setw(probel + 10) << std::setfill(' ');
            ((AAA*)storage[u])->print();
            if (u == y)
            {    std::cout << '\n';
                 if (y == 0)
                       y = 2;
                 else
                       y += y * 2;
            }
            probel /= 2;
```

```
                }
                std::cout << '\n';
            }
        }
```

3. В правой части данного пункта представлена *главная функция* и функция сравнения.

Дополнить комментарии к программе.

```
#include "stdafx.h"
#include "Heap.h"
#include <iostream>
using namespace std;
heap::CMP cmpAAA(void*  a1, void* a2)  //Функция сравнения
{
#define A1 ((AAA*)a1)
#define A2 ((AAA*)a2)
      heap::CMP
      rc = heap::EQUAL;
      if (A1->x  >  A2->x)
            rc = heap::GREAT;
      else
            if (A2->x  > A1->x)
                  rc = heap::LESS;
      return rc;
#undef A2
#undef A1
}
//-------------------------------
int _tmain(int argc, _TCHAR* argv[])
{     setlocale(LC_ALL, "rus");
      int k, choice;
      heap::Heap h1 = heap::create(30, cmpAAA);
      for (;;)
      {     cout << "1 - вывод кучи на экран" << endl;
```

```
                cout << "2 - добавить элемент" << endl;
                cout << "3 - удалить максимальный элемент" << endl;
                cout << "0 - выход" << endl;
                cout << "сделайте выбор" << endl;  cin >> choice;
                switch (choice)
                {     case 0:  exit(0);
                      case 1:  h1.scan(0);
                               break;
                      case 2:  { AAA *a = new AAA;
                                 cout << "введите ключ" << endl; cin >> k;
                                 a->x = k;
                                 h1.insert(a);
                               }
                               break;
                      case 3:   h1.extractMax();
                                break;
                      default:  cout << endl << "Введена неверная команда!" << endl;
                }
          } return 0;
    }
```

5. В проект добавить следующие функции: удаление минимального **extractMin**; удаление i-ого элемента **extractI**; объединение **unionHeap** двух куч в одну.

Тест "Бинарные кучи"

В начало практикума

Лабораторная работа № 14. Хеш-таблицы с открытой адресацией

Хеш-табли́ца (перемешанная таблица) – это структура данных, которая позволяет хранить пары (ключ, значение) и выполнять три операции: добавления новой пары, поиска и удаления пары по ключу. Основное отличие таблиц от других динамических множеств – **вычисление адреса** элемента по значению ключа с помощью хеш-функции. Ситуация, когда для различных ключей получается одно и то же значение хеш-функции, называется *коллизией*.

Существуют два основных варианта хеш-таблиц: *с цепочками* (хеш-таблица содержит списки пар) и *открытой адресацией* (хеш-таблица представляет собой некоторый массив, элементы которого есть пары).

Задание	Краткие теоретические сведения
1. Пункты 1, 2, 3 содержат программный код *проекта*, в котором реализована хеш-таблица с *открытой адресацией*. В правой части данного пункта записан *заголовочный* файл **Hash.h**. Написать комментарии к программному коду.	`#pragma once` `#define HASHDEL (void*) -1` `struct Object` `{ void** data;` `Object(int, int(*)(void*));` `int size; int N;` `int(*getKey)(void*);` `bool insert(void*);` `int searchInd(int key);` `void* search(int key);` `void* deleteByKey(int key);` `bool deleteByValue(void*);` `void scan(void(*f)(void*));` `};` `static void* DEL = (void*)HASHDEL;` `Object create(int size, int(*getkey)(void*));` `#undef HASHDEL`

2. В правой части данного пункта представлена структура, *главная функция* проекта и две вспомогательных функции.

Написать комментарии к программе.

```
#include "stdafx.h"
#include "Hash.h"
#include <iostream>
using namespace std;
struct AAA
{     int key;
      char *mas;
      AAA(int k, char*z)
      {     key = k;  mas = z;
      } AAA() {}
};
//-------------------------------
int key(void* d)
{     AAA* f = (AAA*)d;   return f->key; }
//-------------------------------
void AAA_print(void* d)
{     cout << " ключ " << ((AAA*)d)->key << " - " << ((AAA*)d)->mas << endl; }
//-------------------------------
int _tmain(int argc, _TCHAR* argv[])
{     setlocale(LC_ALL, "rus");
      AAA a1(1, "one"), a2(2, "two"), a3(4, "three"), a4(2, "fo");
      int siz = 10, choice, k;
      cout << "Введите размер хеш-таблицы" << endl;    cin >> siz;
      Object H = create(siz, key);
      for (;;)
      {     cout << "1 - вывод хеш-таблицы" << endl;
            cout << "2 - добавление элемента" << endl;
            cout << "3 - удаление элемента" << endl;
            cout << "4 - поиск элемента" << endl;
            cout << "0 - выход" << endl;
```

```
                cout << "сделайте выбор" << endl;   cin >> choice;
                switch (choice)
                {     case 0:  exit(0);
                      case 1: H.scan(AAA_print);  break;
                      case 2: { AAA *a = new AAA;
                                char *str = new char[20];
                                cout << "введите ключ" << endl; cin >> k;
                                a->key = k;
                                cout << "введите строку" << endl; cin >> str;
                                a->mas = str;
                                if (H.N == H.size)
                                      cout << "Таблица заполнена" << endl;
                                else
                                      H.insert(a);
                               } break;
                      case 3:  {  cout << "введите ключ для удаления" << endl;
                                  cin >> k;
                                  H.deleteByKey(k);
                                } break;
                      case 4:  {  cout << "введите ключ для поиска" << endl;
                                  cin >> k;
                                  if (H.search(k) == NULL)
                                     cout << "Элемент не найден" << endl;
                                  else
                                     AAA_print(H.search(k));
                                } break;
                }
          }
          return 0;
    }
```

3. В правой части записан программный модуль **Hash.cpp**.

Написать комментарии к программному коду.

Сформировать программный код в один проект и выполнить его.

```
#include "stdafx.h"
#include "Hash.h"
#include <iostream>
int HashFunction(int key, int size, int p)    //Хеш-функция
{    double key2=5 * ((0.6180339887499 * key) - int((0.6180339887499 * key)));
     return (p + key) % size;
}
//------------------------------
int Next_hash(int hash, int size, int p)
{    return (hash + 5 * p + 3 * p * p) % size; }
//------------------------------
Object create(int size, int(*getkey)(void*))
{    return *(new Object(size, getkey)); }
//------------------------------
Object::Object(int size, int(*getkey)(void*))
{    N = 0;
     this->size = size;
     this->getKey = getkey;
     this->data = new void*[size];
     for (int i = 0; i < size; ++i)
          data[i] = NULL;
}
//------------------------------
bool Object::insert(void* d)
{    bool b = false;
     if (N != size)
          for (int i = 0, t = getKey(d), j = HashFunction(t, size, 0);
i != size && !b;  j = Next_hash(j, size, ++i))
               if (data[j] == NULL || data[j] == DEL)
               {    data[j] = d;
```

```
                        N++;
                        b = true;
                    }
        return b;
    }
    //-------------------------------
    int Object::searchInd(int key)
    {     int t = -1;
          bool b = false;
          if (N != 0)
             for (int i = 0, j = HashFunction(key, size, 0); data[j] != NULL && i
    != size && !b;  j = HashFunction(key, size, ++i))
                if (data[j] != DEL)
                    if (getKey(data[j]) == key)
                    {     t = j; b = true;      }
          return t;
    }
    //-------------------------------
    void* Object::search(int key)
    {     int t = searchInd(key);
          return(t >= 0) ? (data[t]) : (NULL);
    }
    //-------------------------------
    void* Object::deleteByKey(int key)
    {     int i = searchInd(key);
          void* t = data[i];
          if (t != NULL)
          {     data[i] = DEL;
                N--;
          }
```

```
        return t;
}
//------------------------------
bool Object::deleteByValue(void* d)
{    return(deleteByKey(getKey(d)) != NULL); }
//------------------------------
void Object::scan(void(*f)(void*))
{    for (int i = 0; i < this->size; i++)
     {    std::cout << " Элемент" << i;
          if ((this->data)[i] == NULL)
               std::cout << "  пусто" << std::endl;
          else
               if ((this->data)[i] == DEL)
                         std::cout << "  удален" << std::endl;
               else
                    f((this->data)[i]);
     }
}
```

4. В соответствии со своим вариантом построить хеш-таблицы разного размера (например, 16, 32 или 32, 64, 128) с коллизиями. Исследовать время поиска в хеш-таблицах. В приложении Excel построить соответствующие графики.

№ варианта	Условие задачи
1	Изменить функцию вычисления хеш для решения коллизии на квадратичную функцию, которая строится на основе формулы: **h(key, i) = (h'(key) + c_1*i+ c_2*i^2) mod hashTableSize**.
2	Использовать в проекте функцию универсального хеширования.

№ вари-анта	Условие задачи
3	Изменить функцию вычисления хеш на мультипликативную функцию, которая строится на основе формулы: **H(key) = [hashTableSize(key · A mod 1)]**, где **A = (sqrt(5) - 1) / 2 = 0,6180339887499**.
4	Использовать в проекте функции универсального хеширования и модульного. Сравнить количество коллизий при введении одинаковых ключей.
5	Использовать в проекте линейный алгоритм вычисления последовательности испробованных мест при открытой адресации.
6	Использовать в проекте функции мультипликативного и модульного хеширования. Сравнить время поиска информации.
7	Изменить функцию вычисления хеш на универсальную.
8	Изменить проект с тем, чтобы использовался аддитивный метод хеширования (ключи должны быть строковыми данными).
9	Изменить хеш-функцию в проекте на модульную.
10	Изменить функцию вычисления хеш для решения коллизии на линейную функцию, которая строится на основе формулы: **h(key, i) = (h'(key) + i)·mod hashTableSize**.
11	Написать функцию для доступа к последовательным элементам хеш-таблицы в несортированном порядке.
12	Реализовать динамическую хеш-таблицу с открытой адресацией для хранения строк. Таблица должна увеличивать свой размер вдвое при достижении 50% заполнения.
13	Использовать в проекте функции мультипликативного и модульного хеширования. Сравнить количество коллизий при введении одинаковых ключей.

№ вари-анта	Условие задачи
14	Использовать в проекте функции универсального хеширования и модульного. Сравнить время поиска информации.
15	Изменить функцию вычисления хеш для решения коллизии на функцию, которая строится на основе формулы: **h(key, i) = (h1(key) + i·h2(key))·mod hashTableSize**, где **h1 (key) = key·mod hashTableSize** , **h2(key) = 1 + (key·mod hashTableSize)**.
16	Реализовать хеш-таблицу с открытой адресацией для хранения строк. Таблица должна увеличивать свой размер втрое при достижении 70% заполнения.

Здесь **h'(key)** – значение хеш-функции, приведшее к коллизии.

В начало практикума

Лабораторная работа № 15. Хеш-таблицы с цепочками

В хеш-таблицах с цепочками (с прямой адресацией) каждая ячейка массива является указателем на связный *список* (цепочку) пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии здесь приводят к тому, что появляются цепочки длиной более одного элемента.

Задание	Краткие теоретические сведения
1. Пункты 1, 2, 3, 4 содержат программный код проекта, в котором реализована хеш-таблица *с цепочками*. В правой части записан *заголовочный* файл **Hash_Twin_Chain.h** и *заголовочный* файл **Lists.h**. Написать комментарии к программному коду.	

```
#pragma once                //Заголовочный файл Hash_ Twin_Chain.h
#include "Lists.h"
namespace hashTC
{     struct Object
      {     int size;
            int(*FunKey)(void*);
            listx::Object* Hash;
            Object(int size, int(*f)(void*))
            {     size = size;
                  FunKey = f;
                  Hash = new listx::Object[size];
            };
            int hashFunction(void* data);
            bool insert(void* data);
            listx::Element* search(void* data);
            bool deleteByData(void* data);
            void Scan();
      };
      Object create(int size, int(*f)(void*));
}
```

```
#pragma once              //Заголовочный файл Lists.h
#define LISTNIL (Element*)-1
namespace listx
{     struct Element
      {     Element* prev;
            Element* next;
            void* data;
            Element(Element* prev, void* data, Element* next)
            {     prev = prev;
                  data = data;
                  next = next;
            }
            Element* getNext()
            { return next; };
            Element* getPrev()
            { return prev; };
      };
      static Element* NIL = NULL;
      struct Object
      {     Element* head;
            Object()
            { head = NIL; };
            Element* getFirst()
            { return head; };
            Element* getLast();
            Element* search(void* data);
            bool insert(void* data);
            bool deleteByElement(Element* e);
            bool deleteByData(void* data);
```

```
        void scan();
    };
    Object create();
}
#undef LISTNIL
```

2. В правой части данного пункта записаны функции проекта в файле **Hash_Table.cpp**.

```
#include "stdafx.h"
#include "Hash_Twin_Chain.h"
#include "Lists.h"
#include <iostream>
struct AAA
{   int key;
    char *mas;
    AAA(int k, char*z)
    {   key = k;
        mas = z;
    }
};
namespace hashTC
{   Object create(int size, int(*f)(void*))
    { return *(new Object(size, f)); }
    int Object::hashFunction(void* data)
    { return (FunKey(data) % size); };
    bool Object::insert(void* data)
    { return (Hash[hashFunction(data)].insert(data)); };
    bool Object::deleteByData(void* data)
    { return (Hash[hashFunction(data)].deleteByData(data)); };
    listx::Element* Object::search(void* data)
    { return Hash[hashFunction(data)].search(data); };
    void Object::Scan()
```

```
        {     for (int i = 0; i < size; i++)
              {     Hash[i].scan();
                    std::cout << '\n';
              }
        };
  }
```

3. В правой части записаны функции проекта в файле **Lists.cpp**

```
  #include "stdafx.h"
  #include "Lists.h"
  #include <iostream>
  struct AAA        //элемент таблицы
  {     int key;
        char *mas;
  };
  namespace listx
  {     bool Object::insert(void* data)
        {     bool rc = NULL;
              if (head == NULL)
                    head = new Element(NULL, data, head);
              else
                    head = (head->prev = new Element(NULL, data, head));
              return rc;
        }
  //-------------------------------
        Element* Object::search(void* data)
        {     Element* rc = head;
              while ((rc != NULL) && ((((AAA*)rc->data)->key) != ((AAA*)data)->key))
                    rc = rc->next;
              return rc;
        }
```

```
//-------------------------------
     bool Object::deleteByElement(Element* e)
     {    bool rc = NULL;
          if (rc = (e != NULL))
          {    if (e->next != NULL)
                    e->next->prev = e->prev;
               if (e->prev != NULL)
                    e->prev->next = e->next;
               else
                    head = e->next;
               delete e;
          }
          std::cout << "Элемент удален" << std::endl;
          return rc;
     }
//-------------------------------
     bool Object::deleteByData(void* data)
     {    return deleteByElement(search(data)); }
//-------------------------------
     Element* Object::getLast()
     {    listx::Element* e = this->getFirst(), *rc = this->getFirst();
          while (e != NULL)
          {    rc = e;
               e = e->getNext();
          };
          return rc;
     }
     Object create()
     {    return *(new Object());    };
//-------------------------------
```

```
void Object::scan()
{    listx::Element* e = this->getFirst();
     while (e != NULL)
     {    std::cout << ((AAA*)e->data)->key << '-' << ((AAA*)e->data)->mas
<< " / ";
          e = e->getNext();
     };
}
}
```

4. В правой части данного пункта записаны две вспомогательные функции проекта и *главная функция*.

Выполнить проект и проанализировать результаты. Написать комментарии.

```
#include "stdafx.h"
#include "Hash_Twin_Chain.h"
#include <iostream>
using namespace std;
struct AAA
{    int key;
     char *mas;
     AAA(int k, char*z)
     {    key = k;
          mas = z;
     }
     AAA()
     {    key = 0;
          mas = "";
     }
};
//------------------------------
int hf(void* d)
{    AAA* f = (AAA*)d;
```

```
        return f->key;
}
//-------------------------------
void AAA_print(listx::Element* e)
{    std::cout << ((AAA*)e->data)->key << '-' << ((AAA*)e->data)->mas << " / ";
}
//-------------------------------
int _tmain(int argc, _TCHAR* argv[])
{    setlocale(LC_ALL, "rus");
     int current_size = 7;
     cout << "Введите размер хеш-таблицы" << endl;
     cin >> current_size;
     hashTC::Object H = hashTC::create(current_size, hf);
     int choice;
     int k;
     for (;;)
     {    cout << "1 - вывод хеш-таблицы" << endl;
          cout << "2 - добавление элемента" << endl;
          cout << "3 - удаление элемента" << endl;
          cout << "4 - поиск элемента" << endl;
          cout << "0 - выход" << endl;
          cout << "сделайте выбор" << endl;
          cin >> choice;
          switch (choice)
          {
          case 0:  exit(0);
          case 2:  {   AAA *a = new AAA;
                       char *str = new char[20];
                       cout << "введите ключ" << endl;
                       cin >> k;
```

```
                    a->key = k;
                    cout << "введите строку" << endl;
                    cin >> str;
                    a->mas = str;
                    H.insert(a);
                }
                break;
            case 1: H.Scan();
                break;
            case 3: {   AAA *b = new AAA;
                    cout << "введите ключ" << endl;
                    cin >> k;
                    b->key = k;
                    H.deleteByData(b);
                }
                break;
            case 4: AAA *c = new AAA;
                cout << "введите ключ" << endl;
                cin >> k;
                c->key = k;
                if (H.search(c) == NULL)
                     cout << "Элемент не найден" << endl;
                else
                {   cout << "Первый элемент с данным ключом" << endl;
                    AAA_print(H.search(c));
                    cout << endl;
                }
                break;
            }
        }
```

```
        return 0;
    }
```

5. В соответствии со своим вариантом разработать проект для условия из таблицы, представленной ниже. Построить хеш-таблицы разного размера с коллизиями.

Для вариантов с 1 по 8 вычисление хеш-функции произвести по методу универсального хеширования. Для вариантов с 9 по 16 при вычислении хеш-функции использовать алгоритм на основе исключающего ИЛИ для поля строки данных.

Исследовать время поиска информации.

№ варианта	Задание
1	**Техника**. Реализовать хеш-таблицу со следующими полями: название товара, год выпуска. Ключ – год выпуска.
2	**Список электронных адресов.** Создать хеш-таблицу со следующими полями: адрес, фамилия абонента, год создания. Ключ – год создания.
3	**Школа**. Создать хеш-таблицу со следующими полями: номер школы, фамилия директора. Ключ – номер школы.
4	**Библиотека**. Создать хеш-таблицу со следующими полями: номер книги, название книги. Ключ – номер книги. Номер книги – случайное число.
5	**Прайс-лист**. Создать хеш-таблицу со следующими полями: стоимость товара, название товара. Ключ – стоимость товара.
6	**Железная дорога**. Создать хеш-таблицу со следующими полями: номер маршрута, маршрут.

	Ключ – номер маршрута.
7	**Телефонный справочник**. Создать хеш-таблицу со следующими полями: номер телефона, владелец. Ключ – номер телефона. Телефон хранится в переменной типа int, INT_MAX = 32767.
8	**Аэропорт**. Создать хеш-таблицу со следующими полями: номер рейса, аэропорт назначения. Ключ – номер рейса.
9	**Паспорт**. Реализовать хеш-таблицу со следующими полями: номер паспорта, имя клиента. Ключ – номер паспорта.
10	**Клиенты банка**. Реализовать хеш-таблицу со следующими полями: номер счета, ФИО. Ключ – номер счета.
11	**Города**. Реализовать хеш-таблицу со следующими полями: название, численность населения. Ключ – численность населения.
12	**Жильцы**. Реализовать хеш-таблицу со следующими полями: № квартиры, жильцы. Ключ — номер квартиры.
13	**Студенты**. Реализовать хеш-таблицу со следующими полями: № группы, количество студентов. Ключ – номер группы.
14	**Университеты**. Реализовать хеш-таблицу со следующими полями: год создания, название университета, ФИО ректора. Ключ – год создания.
15	**Паркинг**. Реализовать хеш-таблицу со следующими полями: № места, владельца места. Ключ – № места.
16	**Машины**. Реализовать хеш-таблицу со следующими полями: № машины, владелец машины, марка машины. Ключ – № машины.

6. Дополнительные задания.

1. Составить хеш-таблицу, содержащую буквы и количество их вхождений во введенной строке. Вывести таблицу на экран. Осуществить поиск введенной буквы в хеш-таблице.

2. Построить хеш-таблицу из слов произвольного текстового файла, задав ее размерность с экрана. Вывести построенную таблицу слов на экран. Осуществить поиск введенного слова. Выполнить программу для различных размерностей таблицы и сравните количество сравнений. Удалить все слова, начинающиеся на указанную букву, выведите таблицу.

3. Построить хеш-таблицу для зарезервированных слов, используемого языка программирования (не менее 20 слов), содержащую HELP для каждого слова. Выдатье на экран подсказку по введенному слову. Добавить подсказку по вновь введенному слову, используя при необходимости реструктуризацию таблицы. Сравнить эффективность добавления ключа в таблицу или ее реструктуризацию для различной степени заполненности таблицы.

4. В текстовом файле содержатся целые числа. Построить хеш-таблицу из чисел файла. Осуществить поиск введенного целого числа в хеш-таблице. Сравнить результаты количества сравнений при различном наборе данных в файле.

Тест "Хеш-таблицы"

В начало практикума

Лабораторная работа № 16. Сортировка массивов

Сортировка – это процесс упорядочения набора элементов в возрастающем или убывающем порядке. В рассматриваемых алгоритмах будем считать, что целые положительные и отрицательные числа сортируются по *возрастанию*.

Задание	Краткие теоретические сведения
1. Выполнить программу *сортировки пузырьком*, записанную в правой части. Опробовать ее работу с различными массивами чисел. Заменить функцию пузырьковой сортировки на *шейкерную*. Изучить отличия алгоритмов.	При **обменной сортировке пузырьком** соседние элементы некоторой последовательности чисел сравниваются между собой. Если первый элемент больше второго, то они меняются местами. Затем сравниваются второй и третий элементы. Процесс продолжается до тех пор, пока все элементы массива не окажутся на своих местах. `#include <iostream>` `using namespace std;` `void bubbleSort(int a[], int n)` `{` `  int i, j, t;` `  for (i = 1; i < n; i++)` `  for (j = n - 1; j >= i; j--)` `  if (a[j - 1] > a[j])` `  { t = a[j - 1];` `  a[j - 1] = a[j];` `  a[j] = t;` `  }` `}`

```
void main()
{
	setlocale(LC_ALL, "Rus");
	int size, i;
	int A[100];
	cout << "Количество элементов = ";
	cin >> size;
	for (i = 0; i < size; i++)
	{
		cout << i + 1 << " элемент = ";
		cin >> A[i];
	}
	bubbleSort(A, size);
	cout << "Результирующий массив: ";
	for (i = 0; i < size; i++)
		cout << A[i] << " ";
	cout << endl;
}
```

Перестановка элементов в **шейкерной** сортировке выполняется аналогично пузырьковой, т. е. два соседних элемента при необходимости меняются местами.

При этом можно всякий последующий просмотр делать в противоположном направлении и фиксировать нижнюю и верхнюю границу в неупорядоченной части. Просмотр будет выполняться не до конца массива, а до последней перестановки на предыдущем просмотре.

2. В правой части приведена функция сортировки *выбором*.

Написать программу с ее использованием.

```
void selectSort(int A[], int size)
{     int k, i, j;
      for (i = 0; i < size - 1; i++)
      {     for (j = i + 1, k = i; j < size; j++)
                  if (A[j] < A[k])
                        k = j;
            int c = A[k];
            A[k] = A[i];
            A[i] = c;
      }
}
```

Согласно методу **сортировки выбором** начиная с первой записи таблицы, осуществляется поиск элемента, имеющего наименьшее значение. После того, как этот элемент найден, он меняется местами с первым элементом.

Затем, начиная со второго элемента массива, осуществляется поиск следующего наименьшего значения элемента. Найденный элемент меняется местами со вторым элементом.

Этот процесс продолжается до тех пор, пока все числа не будут расположены в порядке возрастания.

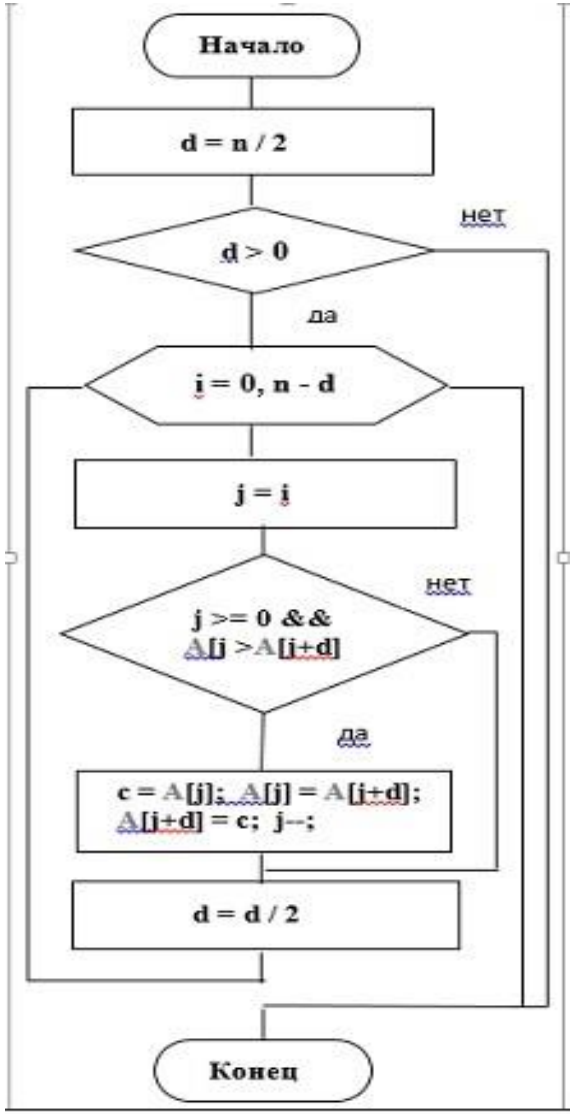
3. В правой части приведена функция сортировки *вставками* и блок-схема программы сортировки методом Шелла **(**сортировка вставками с убывающим шагом).

Написать программы с использованием этих функций и сравнить скорости получения результатов.

```
void insertSort(int *A, int size)
{
    int t, i, j;
    for (i = 1; i < size; i++)
    {
       t = A[i];
       for (j = i-1; j >= 0 && A[j] > t; j--)
            A[j + 1] = A[j];
       A[j + 1] = t;
    }
}
```

При использовании метода **простой вставки** весь массив в процессе сортировки делится на две части: упорядоченную и неупорядоченную.

Вначале весь массив неупорядочен. На каждом шаге из неупорядоченной части извлекается первый элемент, который вставляется на нужное место упорядоченной части. При этом размер упорядоченной части увеличивается на единицу. В конце весь массив окажется упорядоченным.



Метод **Шелла (**сортировка вставками с убывающим шагом) состоит в том, что упорядочиваемый массив делится на группы элементов, каждая из которых упорядочивается методом вставки. В процессе сортировки размеры таких групп увеличиваются до тех пор, пока все элементы не войдут в упорядоченную группу.

4. Разработать главную функцию, реализующую алгоритм *быстрой сортировки Хоара* с использованием функций, приведенных в правой части.

```
int getHoarBorder(int A[], int sm, int em)
{
	int i = sm - 1, j = em + 1;
	int brd = A[sm];
	int buf;
	while (i < j)
	{
		while (A[--j]> brd);
		while (A[++i]< brd);
		if (i < j)
		{
			buf = A[j];
			A[j] = A[i];
			A[i] = buf;
		};
	}
	return j;
}
int* sortHoar(int A[], int sm, int em)
{
	if (sm < em)
	{
	    int hb = getHoarBorder(A, sm, em);
	    sortHoar(A, sm, hb);
	    sortHoar(A, hb + 1, em);
	}
	return A;
};
```

Алгоритм **сортировки разделением** основан на разбиении массива на две части по некоторому правилу.

Быстрая сортировка была разработана **Хоаром** и представляет собой рекурсивный алгоритм. Массив делится на две части относительно некоторого значения, называемого *медианой*. В левой части числа меньше медианы, в правой – больше.

Эти части сортируются независимо, для чего вызывается та же самая функция. Процесс продолжается до тех пор, пока очередная часть массива не станет содержать один элемент.

Функция

getHoarBorder(int A[], int sm, int em); осуществляет разбиение массива.

Здесь **A[]** – исходный массив чисел, **sm** – индекс первого элемента массива, **em** – индекс последнего (последний элемент правой части).

5. В соответствии со своим вариантом написать программу сортировок массивов указанными в таблице методами. Исходные массивы заполняются случайными числами. Определить зависимость времени выполнения алгоритмов от количества элементов для каждого из алгоритмов. Выполнить моделирование для массивов различных размеров. Произвести сравнение эффективности алгоритмов (построить график в приложении Excel).

№ варианта	**Методы сортировки массивов**
1	Ввести массив **A,** в массив **B** скопировать все элементы массива **A**, имеющие четный индекс. Массив **B** отсортировать по убыванию, используя алгоритмы сортировок: «пузырек», сортировка простой вставкой.
2	Ввести массив **A**, в массив **B** перенести все элементы массива **A**, имеющие четный индекс, справа от которых расположены элементы с нечетным значением. Массив **B** отсортировать по убыванию, используя алгоритмы сортировок: сортировка выбором, сортировка Хоара.
3	Ввести массив **A**, в массив **B** перенести все элементы массива **A**, стоящие правее максимального элемента и имеющие нечетный индекс. Массив **B** отсортировать по убыванию, используя алгоритмы сортировок: «пузырек», сортировка Хоара.
4	Ввести массивы **A** и **B**. В массив **C** перенести те элементы массива **A**, которые больше минимального элемента массива **B**. Массив **C** отсортировать по убыванию, используя алгоритмы сортировок: шейкерная сортировка, сортировка Шелла.
5	Ввести массивы **A** и **B**. В массив **C** перенести элементы массива **A** с четным значением и элементы массива **B** с нечетным значением. Массив **C** отсортировать по возрастанию, используя алгоритмы сортировок: «пузырек», сортировка выбором.
6	Ввести массив **A**, в массив **B** скопировать все элементы массива **A**, имеющие четное значение. Массив **B** отсортировать по возрастанию, используя алгоритмы сортировок: сортировка Шелла, сортировка выбором.

№ варианта	Методы сортировки массивов
7	Ввести массивы **A** и **B**. В массив **C** перенести четные элементы массива **A** и нечетные элемента массива **B**. Массив **C** отсортировать по убыванию, используя алгоритмы сортировок: «пузырек», сортировка Хоара.
8	Ввести массивы **A** и **B**. В массив **C** перенести элементы массива **A** с нечетным значением и элементы массива **B** с нечетным значением. Массив **C** отсортировать по возрастанию, используя алгоритмы сортировок: сортировка выбором, сортировка Шелла.
9	Ввести массив **A**, в массив **B** перенести все элементы массива **A**, имеющие нечетный индекс, справа от которых расположены элементы с нечетным значением. Массив **B** отсортировать по возрастанию, используя алгоритмы сортировок: «пузырек», шейкерная сортировка.
10	Ввести массивы **A** и **B**. В массив **C** перенести те элементы массива **A**, которые больше максимального элемента массива **B**. Массив **C** отсортировать по убыванию, используя алгоритмы сортировок: «пузырек», в сортировка простой вставкой.
11	Ввести массивы **A** и **B**. В массив **C** перенести элементы массива **A** с нечетным значением и элементы массива **B** с четным значением. Массив **C** отсортировать по возрастанию, используя алгоритмы сортировок: сортировка Шелла, сортировка Хоара.
12	Ввести массив **A**. В массив **B** скопировать элементы массива **A**, имеющие четный индекс. Массив **B** отсортировать по возрастанию, используя алгоритмы сортировок: «пузырек», сортировка выбором.
13	Ввести массив **A**, в массив **B** перенести все элементы массива **A**, стоящие правее максимального элемента и имеющие четный индекс. Массив **B** отсортировать по убыванию, используя алгоритмы сортировок: «пузырек», сортировка Хоара.
14	Ввести массивы **A** и **B**. В массив **C** перенести нечетные элементы массива **A** и четные элемента массива **B**. Массив **C** отсортировать по убыванию, используя алгоритмы сортировок: сортировка Шелла, «пузырек».

№ варианта	Методы сортировки массивов
15	Ввести массивы **A** и **B**. В массив **C** перенести те элементы массива **A**, которые меньше максимального элемента массива **B**. Массив **C** отсортировать по убыванию, используя алгоритмы сортировок: «пузырек», сортировка Хоара.
16	Ввести массив **A**, в массив **B** перенести все элементы массива **A**, стоящие правее минимального элемента и имеющие нечетный индекс. Массив **B** отсортировать по убыванию, используя алгоритмы сортировок: сортировка Шелла, сортировка Хоара.

Тест "Сортировка массивов"

В начало практикума

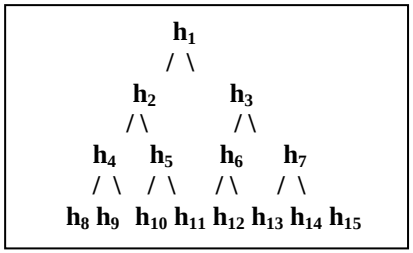
Лабораторная работа № 17. Анализ алгоритмов сортировок

Задание	Краткие теоретические сведения
1. Написать программу, реализующую алгоритм сортировки *подсчетом* с использованием функции, приведенной в правой части. Проанализировать алгоритм.	При использовании **сортировки подсчетом** упорядоченный массив получается из исходного путем сравнения всех пар элементов массива. Для каждого из элементов подсчитывается количество элементов, меньших его. Например, пусть есть массив **B = <20, -5, 10, 8, 7>** . Для первого числа имеется 4 элемента, которые меньше первого. Для второго – 0 элементов и т. д. Таким образом, формируется массив счетчиков **S = < 4, 0, 3, 2, 1**>, элементы которого дают местоположения элементов в результирующем массиве **B' = <-5, 7, 8, 10, 20**>. `void countSort(int in[], int out[], int size)` `{` `  int i, j, count;` `  for (i = 0; i < size; ++i)` `  {` `  for (count = 0, j = 0; j < size; ++j)` `  if (in[j] < in[i] || (in[j] == in[i] && i<j))` `  count++;` `  out[count] = in[i];` `  }` `}` Для сортировки требуются дополнительные массивы для размещения результатов и для счетчиков.

2. Написать программу, реализующую алгоритм *пирамидальной сортировки* с использованием функций, приведенных в правой части.

Проанализировать алгоритм.

Пирамида – это двоичное дерево, в котором значение каждого элемента больше либо равно значениям дочерних элементов. Заполнив дерево элементами в произвольном порядке, можно его отсортировать, превратив в пирамиду. Самый большой элемент пирамиды находится в её вершине. Отделяется вершинный элемент и записывается в конец результирующего массива. На место вершинного элемента помещается элемент из самого нижнего уровня дерева. Восстанавливается (пересортировывается) пирамида. Самый большой элемент из оставшихся элементов снова в вершине и так далее...




```
void heapify(int A[], int pos, int n)
{  int t, tm;
   while (2 * pos + 1 <  n)
   {    t = 2 * pos + 1;
        if (2 * pos + 2 < n  &&  A[2 * pos + 2] >= A[t])
             t = 2 * pos + 2;
        if (A[pos] < A[t])
        {    tm = A[pos];
             A[pos] = A[t];
             A[t] = tm;
             pos = t;
         }
        else break;
   }
}
void piramSort(int A[], int n)
{  for (int i = n - 1; i >= 0; i--)
        heapify(A, i, n);
   while (n > 0)
```

```
    {     int tm = A[0];
          A[0] = A[n - 1];
          A[n - 1] = tm;
          n--;
          heapify(A, 0, n);
    }
}
```

3. Написать программу, реализующую алгоритм сортировки *слиянием* с использованием функций, приведенных в правой части.

Проанализировать алгоритм.

При использовании алгоритма сортировки **слиянием** исходный массив делится пополам, затем рекурсивно упорядочиваются обе половины и объединяются в одно целое. В ходе слияния элементы, стоящие в разных частях массива, попарно сравниваются друг с другом, и меньший элемент отправляется во временный массив. Этот процесс продолжается до тех пор, пока не будет использована полностью одна из двух частей массива. Затем копируются оставшиеся элементы во временный массив, и далее содержимое временного массива копируется обратно в исходный.

```
void insOrd(int m[], int sm, int em, int e)
{ // m[] – массив чисел; sm – индекс 1-ого элемента левой части; cm – индекс
  //последн. элемента левой части; em – индекс последн. элемента правой части
     int buf;
     int i = sm;
     while (i <= em  &&  m[i] <  e)
     {    if (i - 1 >= sm)
               m[i - 1] = m[i];
          i++;
     }  m[i - 1] = e;
 }
int*  merge(int m[], int sm, int cm, int em)
{    for (int i = 0; i <= sm; i++)
     {    if (m[i] > m[cm + 1])
```

```
            {     int buf = m[i];
                  m[i] = m[cm + 1];
                  insOrd(m, cm + 1, em, buf);
            }
      }
      return m;
}
int* sortMerge(int m[], int lm, int sm = 0)
{     if (lm  > 1)
      {     sortMerge(m, lm / 2, sm);
            sortMerge(m, lm - lm / 2, sm + lm / 2);
            merge(m, sm, sm + lm / 2 - 1, sm + lm - 1);
      };
      return m;
}
```

3. В правой части приведена программа, в которой определяется количество временных тактов, требуемых на каждую из двух сортировок.

```
#include <ctime>
#include <stdlib.h>
#include <iostream>
using namespace std;
int* sort1(int m[], int lm)
{     int buf;
      for (int i = 0; i < lm; i++)
            for (int j = 0; j < lm - i - 1; j++)
                  if (m[j] > m[j + 1])
                  {     buf = m[j];
                        m[j] = m[j + 1];
                        m[j + 1] = buf;
                  }
      return m;
}
```

```
//------------------------------
int* sort2(int m[], int lm)
{    int buf;
     bool sort;
     for (int g = lm / 2; g > 0; g /= 2)
          do
          {    sort = false;
               for (int i = 0, j = g; j < lm; i++, j++)
                    if (m[i] > m[j])
                    {    sort = true;
                         buf = m[i];
                         m[i] = m[j];
                         m[j] = buf;
                    }
          } while (sort);
          return m;
}
//------------------------------
int getRandKey(int reg = 0)     // генерация случайных ключей
{    if (reg > 0)
          srand((unsigned)time(NULL));
     int rmin = 0;
     int rmax = 100000;
     return (int)(((double)rand() / (double)RAND_MAX) * (rmax - rmin) + rmin);
}
//------------------------------
int main()
{    setlocale(LC_CTYPE, "Russian");
     const int N = 50000;
     int k[N], f[N];
```

```
        clock_t  t1, t2;
        getRandKey(1);
        for (int i = 0; i < N; i++)
             f[i] = getRandKey(0);
        for (int n = 10000; n <= N; n += 10000)
        {
             cout << "n = " << n << endl;
             cout << "SortPuzirek ";
             memcpy(k, f, n*sizeof(int));
             t1 = clock();
             sort1(k, n);
             t2 = clock();
             cout << "Прошло " << t2 - t1 << " тактов времени" << endl;
             cout << "SortShell   ";
             memcpy(k, f, n * sizeof(int));
             t1 = clock();
             sort2(k, n);
             t2 = clock();
             cout << "Прошло " << t2 - t1 << " тактов времени" << endl << endl;
        }
        return 0;
    }
```

Здесь функция **memcpy(k, f, n*sizeof(int))** осуществляет копирование **n** байтов блока памяти, на который ссылается указатель **f**, во второй блок памяти, на который ссылается указатель **k**.

Функция **clock()** возвращает количество временных тактов, прошедших с начала запуска программы.

4. В соответствии со своим вариантом написать программу для сортировок массивов указанными в таблице методами. Исходные массивы заполняются случайными числами.

Определить зависимость времени выполнения алгоритмов от количества элементов для каждого из алгоритмов. Выполнить моделирование для массивов размером 1000, 2000, 3000, 4000, 5000 (в зависимости от быстродействия компьютера размеры массивов можно увеличивать).

Произвести сравнение эффективности алгоритмов (построить график в приложении Excel).

№ варианта	**Методы сортировки массивов**
1	Сортировка пузырьком, сортировка подсчетом, сортировка Хоара
2	Шейкерная сортировка, пирамидальная сортировка, сортировка слиянием
3	Сортировка выбором, сортировка слиянием, сортировка Шелла
4	Сортировка Хоара, пирамидальная сортировка, сортировка слиянием
5	Сортировка пузырьком, сортировка Шелла, сортировка подсчетом
6	Сортировка методом простой вставки, пирамидальная сортировка, сортировка Хоара
7	Шейкерная сортировка, сортировка слиянием, сортировка Хоара
8	Сортировка Хоара, пирамидальная сортировка, сортировка слиянием
9	Сортировка пузырьком, сортировка выбором, сортировка подсчетом
10	Сортировка выбором, сортировка слиянием, пирамидальная сортировка
11	Сортировка подсчетом, сортировка Хоара, сортировка пузырьком
12	Сортировка Хоара, пирамидальная сортировка, сортировка выбором

№ варианта	Методы сортировки массивов
13	Сортировка пузырьком, сортировка слиянием, сортировка Хоара
14	Шейкерная сортировка, пирамидальная сортировка, сортировка Хоара
15	Сортировка подсчетом, сортировка выбором, сортировка слиянием
16	Сортировка методом простой вставки, сортировка Хоара, пирамидальная сортировка

5. Дополнительные задания.

1. Разработать функцию сортировки квадратичной выборкой и добавить к списку исследуемых сортировок. Алгоритм квадратичной сортировки: массив из **n** элементов делится на группы примерно по **√n** элементов в каждой; в группе выбирается наименьший элемент (его место занимает число, заведомо большее, чем все числа) и пересылается во вспомогательный список; список просматривается, и наименьший его элемент пересылается в окончательный список; просматриваются попеременно, то вспомогательный список, то группы до тех пор, пока все элементы групп не будут исчерпаны.

2. Дан массив целых чисел, количество элементов которого надо ввести с клавиатуры. Найти максимальный элемент массива и его номер, при условии, что все элементы различны. Найти минимальный элемент массива.

3. Дан массив из 10 элементов. Первые 4 элемента упорядочить по возрастанию, последние 4 по убыванию.

4. Дан массив из 15 целых чисел на отрезке [-5; 5]. Упорядочить массив, удалив повторяющиеся элементы.

Тест "Алгоритмы сортировок"

В начало практикума

Приложение 1

Создание очереди с приоритетным включением

При решении некоторых задач возникает необходимость в формировании очередей, порядок включения или выборки элементов из которых определяется *приоритетами* элементов. Приоритет в общем случае может быть представлен числовым значением, которое вычисляется на основании значений каких-либо полей элемента, либо на основании внешних факторов.

В данном примере приоритет заключается в том, что каждый элемент при добавлении его в очередь вставляется в нужное место так, чтобы очередь всегда была упорядочена.

```
#include<iostream>
using namespace std;
struct Item
{       int data;
        Item *next;
};
Item *head, *tail;

bool isNull(void)    //Проверка на пустоту
{       return (head == NULL);
}
void deletFirst()   //Извлечение элемента из начала
{       if (isNull())
             cout << "Очередь пуста" << endl;
        else
        {    Item *p = head;
             head = head->next;
             delete p;
        }
}
```

```
void getFromHead()  //Получение элемента из начала
{       if (isNull())
             cout << "Очередь пуста" << endl;
        else
             cout << "Начало = " << head->data << endl;
}
void insertToQueue(int x)  //Добавление элемента в очередь
{     Item *p = new Item;       //новый указатель
      p->data = x;
      p->next = NULL;
      Item *v = new Item;       //указатель для нового числа
      Item *p1 = new Item;
      Item *p2 = new Item;
      int i = 0;                //флажок
      if (isNull())
           head = tail = p;
      else
      {     p2 = head; p1 = head;
           while (p1 != NULL)  //пока очередь не закончится
           {     if (i == 1)
                {     if (x <= p1->data)  //число меньше, чем в очереди
                     {     v->data = x;
                          v->next = p1;
                          p2->next = v;
                          return;
                     }
                     p2 = p2->next;       // следующее число
                }
                else
                {     if (x <= p1->data)
```

```
                    {     v->data = x;
                          v->next = p1;
                          head = v;
                          return;
                    }
               }
               p1 = p1->next;
               i = 1;
          }
          if (p1 == NULL)
          {     tail->next = p;
               tail = p;
          }
     }
  }
void printQueue()              //Вывод очереди
{     Item *p = new Item;
     if (isNull())
          cout << "Очередь пуста" << endl;
     else
     {     cout << "Очередь = ";
          p = head;
          while (!isNull())
          {     if (p != NULL)
               {     cout << p->data << " "; cout << "->";
                    p = p->next;
               }
               else
               {     cout << "NULL" << endl;
                    return;
```

```
                }
            }
        }
}

void clrQueue()           //Очистка очереди
{    while (!isNull()) deletFirst();
}
int main()
{    setlocale(LC_CTYPE, "Russian");
     int i = 1, choice = 1, z; head = NULL;  tail = NULL;
     while (choice != 0)
     {    cout << "1 - добавить элемент" << endl;
          cout << "2 - получить элемент с начала" << endl;
          cout << "3 - извлечь элемент с начала" << endl;
          cout << "4 - вывести элементы" << endl;
          cout << "5 - очистить очередь" << endl;
          cout << "0 - выход" << endl;
          cout << "Выберите действие  ";  cin >> choice;
          switch (choice)
          {    case 1: cout << "Введите элемент:  "; cin >> z;
                       insertToQueue(z); printQueue();  break;
               case 2: getFromHead(); break;
               case 3: deletFirst();  break;
               case 4: printQueue();  break;
               case 5: clrQueue();    break;
          }
     }
     return 0;
}
```

Задания для выполнения

1. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие – подряд из младших адресов (начало очереди). Приоритет: минимальное значение числового параметра, при совпадении параметров – **LIFO**.

2. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие – подряд из начала очереди. Приоритет: иаксимальное значение числового параметра, при совпадении параметров – **FIFO**.

3. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие – подряд из старших адресов (конец очереди). Приоритет: минимальное значение числового параметра, при совпадении параметров – **LIFO**.

4. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие – подряд из старших адресов (конец очереди). Приоритет: минимальное значение числового параметра, при совпадении параметров – **LIFO**.

5. Разработать функции работы с приоритетной очередью. Постановка запросов в очередь выполняется по приоритету, снятие – подряд из старших адресов (конец очереди). Приоритет: максимальное значение числового параметра, при совпадении параметров – **FIFO**.

В начало практикума

Приложение 2

Разработка проекта с использованием двусвязного списка

В проекте используется двусвязный список для хранения информации о телефонных номерах. Проект состоит из трех частей: главная функция с именем **OAP_List**, файл **List.cpp** с функциями пользователя, заголовочный файл **List.h**.

Задание	Краткие теоретические сведения
1. Создать проект, ввести операторы *главной* функции и изучить структуру функции. Добавить операторы создания списков и операций над ними.	Главная функция **OAP_List** организует ввод и редактирование информации, демонстрирует способы обращения к функциям для работы со списком, описанным в файле **List.cpp**. В программе создается список с именем **L1**. Опробуются операции вставки элементов, удаления, поиска. `#include "stdafx.h"` `#include "List.h"` `#include <iostream>` `using namespace std;` `struct Person` `{  char name[20];` `  int age;` `  Person *next;` `};` `void print(void* b)  //Функция используется при выводе` `{  Person *a = (Person*)b;` `  cout << a->name << "  " << a->age << endl;` `}`

```
int _tmain(int argc, _TCHAR* argv[])
{    setlocale(LC_ALL, "Russian");
     Person a1 = { "Петров", 20 };
     Person a2 = { "Сидоров", 25 };
     Person a3 = { "Гончаров", 47 };
     bool rc;
     Object L1 = Create();   // создание списка с именем L1
     L1.Insert(&a1);
     L1.Insert(&a2);
     L1.Insert(&a3);
     L1.PrintList(print);
     Element *e = L1.Search(&a3);
     Person *aa = (Person*)e->Data;
     cout << "Найден элемент= " << aa->name << endl;
     rc = L1.Delete(&a2);
     if(rc == true) cout<<"Удален элемент"<<endl;
     cout<<"Список:"<<endl;
     L1.PrintList(print);
     return 0;
}
```

2. Добавить в проект файл **List.cpp**. Изучить функции программы.

Написать комментарии к операторам.

```
#include "stdafx.h"
#include "List.h"
bool Object::Insert(void* data)           // Вставка в начало
{  bool rc = 0;
   if (Head == NULL)
   {    Head = new Element(NULL, data, Head);
        rc = true;
   }
   else
```

```
    {     Head = (Head->Prev = new Element(NULL, data, Head));
          rc = true;
    }
    return rc;
}
//-------------------------------------------------------------
Element* Object::Search(void* data)    // Найти заданный элемент
{   Element* rc = Head;
    while ((rc != NULL) && (rc->Data != data))
          rc = rc->Next;
    return rc;
}
//-------------------------------------------------------------
void Object::PrintList(void(*f)(void*))     // Вывод
{   Element* e = Head;
    while (e != NULL)
    {     f(e->Data);
          e = e->GetNext();
    };
}
//-------------------------------------------------------------
void Object::PrintList(void(*f)(void*), Element *e)
{   f(e->Data);
}
//-------------------------------------------------------------
bool  Object::Delete(Element* e)    // Удалить по ссылке
{   bool rc = 0;
    if (rc = (e != NULL))
    {     if (e->Next != NULL)
                e->Next->Prev = e->Prev;
```

```
            if (e->Prev != NULL)
                  e->Prev->Next = e->Next;
            else
                  Head = e->Next;
            delete e;
      }
      return rc;
}
//-------------------------------------------------------------
bool Object::Delete(void* data)      // Удалить по значению
{  return Delete(Search(data));  };
//-------------------------------------------------------------
Element* Object::GetLast()
{  Element* e = Head, *rc = e;
   while (e != NULL)
   {     rc = e;
         e = e->GetNext();
   }
   return rc;
}
//-------------------------------------------------------------
Object Create()
{  return *(new Object()); }
```

3. Добавить в проект заголовочный файл **List.h**.

Выполнить программы проекта.

Заголовочный файл **List.h** содержит описание структуры списка, конструктор списка и прототипы программ обработки списка.

```
#pragma once
struct Element                    // Элемент списка
{  Element* Prev;                 // указатель на предыдущий элемент
```

```
    Element* Next;                  // указатель на следующий элемент
    void*  Data;                    //  данные
    Element(Element* prev, void* data, Element* next)
    {     Prev = prev;
          Data = data;
          Next = next;
    }
    Element* GetNext()              // получить следующий
    {     return Next;
    };
    Element* GetPrev()              // получить предыдущий
    {     return Prev;  };
};
//-------------------------------------------------------------
struct Object                       // Блок управления списком
{   Element* Head;                  // указатель на начало списка
    Object()
    {     Head = NULL;  };
    Element* GetFirst()             // получить первый элемент списка
    {     return Head;  };
    Element* GetLast();             // получить последний элемент списка
    Element* Search(void* data);    // найти  первый элемент по данным
    bool Insert(void* data);        // добавить элемент в начало
    bool InsertEnd(void* data);     // добавить в конец
    bool Delete(Element* e);        // удалить по адресу элемента
    bool Delete(void* data);        // удалить первый по данным
    bool DeleteList();              // очистить список
    void Object::PrintList(void(*f)(void*));
    void Object::PrintList(void(*f)(void*), Element*);
    int Object::CountList();
```

	`` bool Object::DeleteDouble(); }; Object Create(); // создать список ```

Здесь директива препроцессора **#pragma once** применяется для защиты от двойного подключения заголовочных файлов.

4. Дополнить проект, разработав функцию **deleteList** (удаление списка) и функцию **countList** (подсчет числа элементов списка).

В содержимом списка отразить информацию в соответствии со своим вариантом из лабораторной работы № 4. Создать интерфейс в виде меню.

В начало практикума