

Документ подписан простой электронной подписью

Информация о владельце:

ФИО: Ильшат Ринатович Мухаметзянов

Должность: директор

Дата подписания: 13.07.2023 12:35:18

Уникальный идентификатор документа:  
aba80b84033c9ef196388e9ea0434f90a83a40954ba270e84bcb664f02d1d8d0

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение  
высшего образования «Казанский национальный исследовательский

технический  
университет им. А.Н. Туполева-КАИ»  
(КНИТУ-КАИ)  
Чистопольский филиал «Восток»

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ**  
по дисциплине  
**ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ**

Индекс по учебному плану: **Б1.О.12.02**

Направление подготовки: **09.03.01 Информатика и вычислительная техника**

Квалификация: **Бакалавр**

Профиль подготовки: **Вычислительные машины, комплексы, системы и сети**

Типы задач профессиональной деятельности: **проектный, производственно-технологический**

Рекомендовано УМК ЧФ КНИТУ-КАИ

Чистополь  
2023 г.

## Практическое занятие 1

### Функции

Задача 1. Составить определение функции  $F(X) = \sqrt{X}$  с погрешностью не более 0.00001.

Решение. Используем итерационную формулу Ньютона (см. задачу 2.3 б). Для вычисления квадратного корня  $k = 2$ , поэтому

$$Y_0=1, Y_{n+1} = ( Y_n + X / Y_n ) / 2.$$

```
/* Программа 8.1. Функция F(x) = квадратный корень (x) */
#include <math.h>
float F (float x)
{ float y;          /*последнее приближение функции      */
  float ypr;        /*предыдущее приближение функции      */
  const float e=0.00001; /* допустимая погрешность            */
  y = 1;
  do
  { ypr = y;
    y = (ypr + x / ypr) / 2;
  } while (fabs(y-ypr) >= e);
  return y;
}
```

2. Составить определения следующих функций.

а) Расстояние между двумя точками, заданными своими координатами на плоскости.

б)  $\max(x, y)$ .

в)  $\min(x, y)$ .

г)  $\max(a, b, c)$ .

д)  $\min(a, b, c)$ .

е) 1, если заданный символ является латинской буквой, и 0 в противном случае.

ж) 0, если нельзя построить треугольник из трех отрезков заданной длины (каждая сторона треугольника должна быть меньше суммы двух других сторон); 1 - треугольник равносторонний, 2 - равнобедренный, 3 - любой другой.

з) Число Фибоначчи  $f(n)$ , где  $f(0) = 0$ ,  $f(1) = 1$ ,  $f(j) = f(j-1) + f(j-2)$  для целого  $j > 1$ .

и) Наибольший общий делитель натуральных чисел A и B.

к) Количество десятичных цифр заданного целого числа.

л) Значение числа, полученного выписыванием в обратном порядке десятичных цифр заданного натурального числа.

м) Количество единичных битов в двоичной записи заданного числа типа unsigned long (см. задачу 2.44).

н) Ввод и вычисление значения двоичного целого числа.

3. Дана последовательность из 150 действительных чисел. Вычислить сумму и среднее арифметическое значение первых 100 чисел и последующих 50 чисел.

Решение. Составим подпрограмму ввода n чисел и вычисления их суммы s и среднего арифметического значения sa. Входным параметром подпрограммы является n, выходными параметрами - s и sa. Для решения задачи вызовем эту подпрограмму для  $n=100$  и  $n=50$ .

```
/*    Программа 8.3. Сумма и среднее арифметическое    */
#include <math.h>
/* Подпрограмма: s = сумма, sa = среднее арифметич. n чисел */
void sum_sr (int n, float *s, float *sa)
{ float x;          /* текущее число    */
  int j;
  *s=0;
  for (j=0; j<n; j++)
```

```

    { scanf("%f", &x);
      *s = *s + x;
    }
    *sa = *s / n;
}

void main()
{ float sum1, sum2, sr1, sr2;
  sum_sr (100, &sum1, &sr1);
  sum_sr (50, &sum2, &sr2);
  printf ("%f %f %f %f", sum1, sr1, sum2, sr2);
}

```

4. Составить подпрограммы для решения следующих задач.

а) Поменять местами значения переменных  $x$ ,  $y$ .

б) Значения переменных  $x$ ,  $y$ ,  $z$  поменять местами так, чтобы оказалось  $x \geq y \geq z$ .

в) Вычислить длину окружности, площадь круга и объем шара одного и того же заданного радиуса.

г) Вычислить площадь и периметр прямоугольного треугольника по длинам двух катетов.

д) Звездочками '\*' нарисовать на экране прямоугольник с заданной шириной и высотой, например, шириной 7 и высотой 4 символа:

```

*****
*       *
*       *
*       *
*****

```

е) Для квадрата ABCD на плоскости даны координаты  $X_A$ ,  $Y_A$ ,  $X_C$ ,  $Y_C$  диагонально расположенных вершин A и C. Вычислить координаты вершин B и D.

ж) Вывод заданного целого числа в двоичной системе счисления.

з) Вывод заданного целого числа в шестнадцатеричной системе счисления.

5. Пусть  $p$ ,  $q$ ,  $r$  - целочисленные беззнаковые переменные. Составить подпрограммы для решения следующих задач.

а) Упаковка. Поместить младшие 3 бита переменной  $p$  в младшие 3 бита переменной  $r$ , а младшие 6 битов переменной  $q$  - в следующие 6 битов переменной  $r$  без изменения остальных битов.

б) Распаковка. Присвоить младшим битам переменной  $p$  младшие 3 бита переменной  $r$ , а младшим битам переменной  $q$  - следующие 6 битов переменной  $r$ . Остальные биты  $p$  и  $q$  обнулить.

## Практическое занятие 2

### Рекурсивные функции

Задача: Дано  $n$  различных натуральных чисел. Напечатать все перестановки этих чисел

```
#include <conio.h>
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int n,ks;
char in[30];
char pr;
void recurs(int n){
    if (n!=ks/2){
        pr=in[n];
        in[n]=in[(ks-1)-n];
        in[(ks-1)-n]=pr;
        recurs(n+1);
    }
}
void main(){
    clrscr();
    cout << "Введите слово:";
    cin >> in;
    ks=0;
    ks=strlen(in);
    n=0;
    recurs(0);
    cout << "\n" << in;
    getch();
}
```

}

1. Найдите сумму цифр заданного натурального числа, используя рекурсивную подпрограмму.

2. Подсчитать количество цифр в заданном натуральном числе, используя рекурсивную подпрограмму.

3. Описать функцию  $C(m,n)$ , где  $0 \leq m \leq n$ , для вычисления биномиального коэффициента  $C_n^m$  по следующей формуле:

$$C_n^0 = C_n^n = 1; C_n^m = C_{n-1}^m + C_{n-1}^{m-1} \text{ при } 0 < m < n$$

используя рекурсивную подпрограмму

4. Описать рекурсивную функцию  $\text{Root}(f, b, \varepsilon)$ , которая методом деления отрезка пополам находит с точностью  $\varepsilon$  корень уравнения  $f(x) = 0$  на отрезке  $[a, b]$  (считать, что  $\varepsilon > 0$ ,  $a < b$ ,  $f(a) \cdot f(b) < 0$  и  $f(x)$ - непрерывная и монотонная на отрезке  $[a, b]$ ).

5. Описать функцию  $\text{min}(x)$  для определения минимального элемента линейного массива  $x$ , введя вспомогательную рекурсивную функцию  $\text{min1}(k)$ , находящую минимум среди последних элементов массива  $x$ , начиная с  $k$ -го.

6. Описать рекурсивную логическую функцию  $\text{Simm}(S, i, j)$ , проверяющую, является ли симметричной часть строки  $S$ , начинающаяся  $i$ -м и кончающаяся  $j$ -м ее элементами.

7. Составить программу вычисления наибольшего общего делителя двух натуральных чисел, используя рекурсивную подпрограмму.

8. Составить программу нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке, используя рекурсивную подпрограмму. Например, для числа 1234 получаем ответ 4321.

9. Составить программу перевода данного натурального числа в  $p$ -ичную систему счисления ( $2 \leq p \leq 9$ ), используя рекурсивную подпрограмму.

10. Дана символьная строка, представляющая собой запись

натурального числа в  $p$ -ичной системе счисления ( $2 \leq p \leq 9$ ). Составить программу перевода этого числа в десятичную систему счисления, используя рекурсивную подпрограмму.

11. Составить программу вычисления суммы:

$$1! + 2! + 3! + \dots + n! \quad (n \leq 20).$$

Примечание. Тип результата значения функции - long.

12. Составить программу вычисления суммы:

$$2! + 4! + \dots + n! \quad (n \leq 20, n - \text{четное}).$$

Примечание: Тип результата значения функции — long.



## Практическое занятие 3

### Рекурсивные функции

Ввести строки. Сравнить первые десять символов двух строк. Если они одинаковы, объединить две строки, исключив из второй первые десять символов. В случае отличия первых десяти символов скопировать вторую строку в первую. Посчитать длину исходной и полученной первой строки.

#### Вариант решения 1.

```
#include <stdio.h>
#include <string.h>
#include <locale.h>
int main()
{
    setlocale (LC_CTYPE, "Russian");
    char a[200], b[200], rest[200];
    char c1[11], c2[11];
    printf ("\nСтрока 1: ");
    fgets (a, 200, stdin);
    if (a[strlen(a) - 1] == '\n')
        a[strlen(a) - 1] = '\0';
    printf ("\nСтрока 2: ");
    fgets (b, 200, stdin);
    if (b[strlen(a) - 1] == '\n')
        b[strlen(a) - 1] = '\0';
    strncpy(c1, a, 10);
    strncpy(c2, b, 10);
    c1[10] = c2[10] = '\0';
    printf ("\nДлина первой строки до изменения:\t, %d", strlen(a));
    if (strcmp(c1, c2) == 0)
    {
        printf( "\nПервые десять символов совпадают. Копируем вторую
строку в первую\n");
        memcpy (rest, b+10, strlen(b)-9);
    }
}
```

```

    strcat(a, rest);
}
else
{   printf ( "\nПервые десять символов не совпадают. Объединяем
строки\n");
    strcat(a,b);
}
printf( "\nСтрока 1: %s",a);
printf( "\nДлина первой строки после изменения:\t %d", strlen(a));
getchar();
return 0;
}

```

## Вариант решения 2

```

#include <stdio.h>
#include <string.h>
int main()
{
    char a[200], b[200], rest[200];
    char c1[11], c2[11];
    printf ("\nСтрока 1: ");
    fgets (a, 200, stdin);
    if (a[strlen(a) - 1] == '\n')
        a[strlen(a) - 1] = '\0';
    printf ("\nСтрока 2: ");
    fgets (b, 200, stdin);
    if (b[strlen(a) - 1] == '\n')

```

```

    b[strlen(a) - 1] = '\0';
    strncpy(c1, a, 10);
    strncpy(c2, b, 10);
    c1[10] = c2[10] = '\0';
    printf ("\nДлина первой строки до изменения:\t, %d", strlen(a));
    if (strcmp(c1, c2) == 0)
    {
        printf( "\nПервые десять символов совпадают. Копируем вторую строку
в первую\n");
        memcpy (rest, b+10, strlen(b)-9);
        strcat(a, rest);
    }
    else
    {
        printf ( "\nПервые десять символов не совпадают. Объединяем
строки\n");
        strcat(a,b);
    }
    printf( "\nСтрока 1: %s",a);
    printf( "\nДлина первой строки после изменения:\t %d", strlen(a));
    return 0;
}

```

1. Ввести предложение, слова в которых разделены пробелами и запятыми. Распечатать это предложение, удалив из него те слова, которые встретились там более одного раза.
2. Даны две символьные строки, состоящие только из цифр (длина каждой более 10 символов). Считая, что в этих строках находятся очень длинные числа, сформировать третью строку- сумму этих чисел.

3. Дан произвольный текст. Отредактировать текст так, чтобы:
  - между словами был ровно один пробел;
  - предложения в тексте разделялись ровно двумя пробелами.
4. Ввести два предложения и распечатать самые длинные слова, общие для этих предложений. Если нужных слов нет - сообщить об этом.

## Практическое занятие 4

### Структуры

Задача 1: Написать программу поиска сотрудников, имеющих стаж работы в ОВД от 10 до 15 лет, использующая функцию *vvod\_dannyh* () для ввода информации о сотрудниках и функцию *poisk* () для организации указанного поиска.

```
/*    Программа 1. Обработка структур                                */
#include "stdafx.h"
#include <iostream>
using namespace std;
struct sotrudnik {
char Fam [20]; int Nomer_udost; int Stag; };
void vvod_dannyh (sotrudnik *v_point);
void poisk (sotrudnik *p_point);
int main()
{
sotrudnik Sev_ROVD[5], *point;
point=&Sev_ROVD[0];
vvod_dannyh (point);
cout <<endl;
cout << "Rezul'taty poiska:"<<endl;
poisk (point);
return 0;
}
void vvod_dannyh (sotrudnik *v_point)
{
for (int i=0; i<5; i++)
{cout << "Vvedite Familiu=";cin >> v_point->Fam;
cout << "Vvedite Nomer slug. udostovereniay=";
cin >> v_point->Nomer_udost;
```

```

cout << "Vvedite stag slugby v OVD=";
cin >> v_point->Stag;
v_point++;}
}
void poisk (sotrudnik *p_point)
{
int k=0;
for (int i=0; i<5; i++)
{
if (p_point->Stag >= 10 && p_point->Stag <= 15)
{k++;
cout << k << ". " << p_point->Fam << "=";
cout << p_point->Stag << " let";
cout << endl; }
p_point++;
}
if (k==0)
cout << "Sotrudnikov s ukazamnym stagem net!" << endl;
}

```

Задача 2:

а) Реализовать массив структур (*Priem*), содержащий сведения об абитуриентах, поступающих в Вуз. Структура *abiturient* имеет следующие поля: фамилию абитуриента (*Fam*), его имя (*Imay*) и отчество (*Otch*), год рождения (*God\_rogd*), номер аттестата о среднем образовании (*Nomer\_at*), средний балл аттестата (*Sr\_ball*), оценка по трем вступительным экзаменам (*Rus, Math, Inf*), контактный телефон (*Tel*). Размер массива не более 25 элементов.

б) Вывести на экран информацию об абитуриентах, набравших не менее 13 баллов по результатам сдачи вступительных экзаменов. Информацию представить в виде таблицы, имеющей следующие столбцы:

фамилия, имя и отчество абитуриента; сумма баллов, набранная на вступительных экзаменах; средний балл аттестата.

Задача 3:

а) Реализовать массив структур (*Biblio*), содержащий сведения о книгах, находящихся в библиотечном фонде Вуза. Структура *kniga* имеет следующие поля: автор книги (*Fio*), название книги (*Name*), год издания (*God\_izd*), издательство (*Izdat*), количество экземпляров (*Kol\_vo*), название дисциплины, в которой используется книга (*Predmet\_name*), шифр книги (*Kod*). Размер массива не более 25 элементов.

б) Вывести на экран информацию о книгах, имеющихся в библиотеке, не позже 2003 года издания. Информацию представить в виде таблицы, имеющей следующие столбцы: автор книги; название, год издания, количество экземпляров.

## Практическое занятие 5

### Работа с файлами

1. Подсчитать, количество в данном файле  $f$  символов «А».
2. Подсчитать, в данном файле  $f$  количество сочетаний КАИ.
3. Распечатать все строки данного файла, содержащие заданную строку в качестве подстроки. Имя файла и сама строка задаются пользователем с клавиатуры.
4. Реализовать программу, которая определяет какой символ чаще других встречается в заданном файле. Имя файла задается пользователем
5. Реализовать программу, которая определяет сколько имеется строк, состоящих из одного, двух, трех и т.д. символов, содержится в данном файле. Считать, что длина каждой строки - не более 80 символов. Имя файла задается пользователем
6. Реализовать программу, которая определяет самую длинную строку в данном файле. Если таких строк несколько, то в качестве результата вывести первую из них. Имя файла задается пользователем.
7. Имеются два непустых файла. Вывести номер строки и номер символа в строке, в котором имеется отличие содержимое одного файла от содержимого второго файла. Если их содержимое абсолютно одинаково, то вывести на экран 0 и соответствующее сообщение; если один из этих файлов можно считать началом другого, то вывести соответствующее сообщение и  $-n+1, 1$ , где  $n$  - количество строк в более коротком файле. Данные файлы задаются своими именами пользователем с командной строки.
8. Имеется непустой файл, в котором содержится последовательность целых чисел. Имя файла задается пользователем в командной строке.
  - вывести наибольшее из этих чисел;
  - вывести, количество четных чисел, которые содержатся в файле;
  - с) вывести, образуется из этих чисел арифметическая прогрессия;
  - д) вывести, являются ли эти числа возрастающей последовательностью;



е) вывести количество чисел последовательности, которые являются точными квадратами;

9. Написать программу, определяющую, какая из строк чаще других встречается в данном файле.

## Практическое занятие 6

### Функции доступа к файлам

Список функций для работы с файловыми потоками хранится в библиотеке (заголовочном файле) `fstream.h`. Поэтому во всех рассматриваемых ниже фрагментах программ предполагается, что в начале программы есть соответствующая директива `#include`:

```
#include <fstream.h>
```

Создание потока ввода-вывода

Прежде чем начать работать с потоком необходимо его создать. Поток ввода создается инструкцией

```
ifstream <имя потока ввода>;
```

Поток вывода создается инструкцией

```
ofstream <имя потока вывода>;
```

Пример:

```
ifstream input;
```

```
ofstream output;
```

эти инструкции создают поток ввода `input` и поток вывода `output`.

Открытие и закрытие файла

После создания потока его можно подключить к файлу (открыть файл) инструкцией

```
<имя потока>.open (<имя файла>);
```

Здесь `<имя файла>` - текстовая константа или переменная.

Например, для подключения потока ввода `ifstream` с именем `input` к файлу `data.txt` надо выполнить инструкцию

```
input.open ("data.txt");
```

Аналогичная инструкция

```
output.open ("data.txt");
```

подключит поток вывода `output` к файлу `data.txt` – файл подготовлен к записи данных. Важно отметить, что при выполнении оператора подготовки файла к записи, прежние данные из файла `data.txt` будут удалены.

Для отключения потока ввода-вывода от файла надо выполнить инструкцию закрытия файла:

```
<имя потока>.close ();
```

Так, инструкции

```
input.close();
```

```
output.close ();
```

отключают потоки ввода `input` и вывода `output` от файла `data.txt`, к которому они были подключены в предыдущих примерах. При закрытии файла вывода в конец файла записывается метка конца `end_of_file`.

### Обработка ошибок

При выполнении операций над файлами, например, открытие и закрытие файлов, достаточно высока вероятность возникновения ошибочных ситуаций.

Один из простейших способов контроля корректности выполнения файловых операций заключается в вызове функции

```
<имя потока>.fail()
```

например,

```
input.fail()
```

Эта инструкция выполняется как вызов булевской функции, которая возвращает значение `false` (0), если последняя операция с потоком `input` завершилась успешно и возвращает значение `true` (1), если последняя операция с потоком `input` привела к ошибке (например, была попытка открытия несуществующего файла). В случае возникновения ошибки поток может быть поврежден, поэтому работу с ним продолжать нельзя.

Распознать ошибку можно и с помощью перегруженной операции отрицания. Выражение `!<имя потока>` также принимает значение `false` (0), если последняя операция с потоком завершилась успешно и принимает значение `true` (1), если последняя операция с потоком привела к ошибке.

Пример:

```
ifstream input;
```

```
input.open ("data.txt");
```

```
if(!input) exit(1); // завершение работы программы
```

функция `exit()` описана в библиотеке `stdlib.h`.

Чтение-запись символов в файл

После того, как файл открыт для ввода данных, из него можно считывать отдельные символы. Чтение текущего символа из потока ввода выполняется инструкцией

```
<имя потока>.get(<имя переменной>);
```

Например, после выполнения инструкции `input.get(ch);` произойдет следующее: переменной `ch` будет присвоено значение текущего символа (шаг 1), и поток `input` будет подготовлен для чтения следующего символа (шаг 2).

Аналогично после того, как файл открыт для вывода данных, в него можно записывать отдельные символы. Запись символа в поток вывода выполняется инструкцией

```
<имя потока>.put(<имя переменной>);
```

Например, после выполнения инструкции `output.put(ch);` произойдет следующее: в поток `output` будет помещено значение символьной переменной `ch` (шаг 1), и поток `output` будет подготовлен для записи следующего символа (шаг 2). Вместо имени переменной можно указать значение выводимого в поток символа: инструкция `output.put('t');` выводит в поток `output` символ `'t'`.

При чтении данных из файла (из потока ввода, связанного с файлом) необходимо уметь определять конец файла. В C++, как и в Паскале, для этой цели используется функция `eof()`. Логическое выражение `<имя потока>.eof()` принимает значение `true (1)`, если конец файла достигнут и значение `false (0)`, если можно продолжать чтение.

Пример (посимвольное чтение данных из файла и вывод на экран):

```
#include <iostream.h>
```

```
#include <fstream.h>
```

```
#include <stdlib.h>
```

```
void main()
```

```

{
    char ch;
    ifstream input;
    input.open("a:/text.txt");
    if(!input)
        {cout << "\nmistake of open\n"; exit(1);} //Ошибка открытия файла
    while (!input.eof()){
        input.get(ch); //чтение очередного символа
        cout << ch; //вывод символа на экран
    }
    input.close();
}

```

### **Ввод-вывод с преобразованием типов**

Для того чтобы программа могла работать с числовыми данными, которые записаны в текстовом файле, необходимо при вводе выполнять преобразование символьной записи чисел (внешнее представление) в их внутреннее представление в памяти компьютера, т.е. выполнять преобразование типов данных при вводе. Аналогично при выводе на экран числа должны преобразовываться из внутреннего (двоичного) во внешнее представление. В Паскале такое преобразование автоматически выполнялось операторами ввода-вывода (**read/write**).

В C++ преобразование числовых данных из внешнего (символьного) представления во внутреннее (двоичное) выполняет оператор **>>**. Обратное преобразование выполняет оператор **<<**. Эти операторы мы уже использовали при работе со стандартными потоками ввода-вывода **cin** и **cout**.

Инструкция ввода данных из потока ввода с преобразованием типов выглядит так:

```
поток ввода >>переменная >>переменная ... >>переменная;
```

Например,

```
input >>a >>b >>c;
```

Аналогично выглядит инструкция вывода данных из потока вывода с преобразованием типов:

```
поток вывода <<выражение <<выражение ... <<выражение;
```

Например,

```
output <<a <<b+c <<2*m[5];
```

Пример (вывод-ввод в файл числовой последовательности):

```
#include <iostream.h>
```

```
#include <fstream.h>
```

```
#include <stdlib.h>
```

```
void main()
```

```
{
```

```
    int n, a;
```

```
    cout <<"\ninput n: "; cin >>n;
```

```
    ifstream input;
```

```
    ofstream output;
```

```
    output.open ("D:/data.txt");
```

```
    if(!input)
```

```
        {cout <<"\nmistake of open\n"; exit(1);} //Ошибка открытия файла
```

```
    for (int i=1; i<=n; i++){
```

```
        cout <<"\ninput next number: "; cin >>a;
```

```
        output <<a;
```

```
        if (i!=n) output <<' '; // output.put(' ')
```

```
    }
```

```
    output.close ();
```

```
    input.open ("D:/data.txt");
```

```
    input >>a;
```

```
    while (!input.eof()){
```

```
        cout <<a <<' '; input >>a;
```

```
    }
```

```
input.close ();  
}
```

При выводе в текстовый файл чисел необходимо предусмотреть разделитель. В приведенном примере в качестве разделителя в файл выводится пробел.

### Чтение символьных строк из потока ввода

Для ввода строк из потока ввода (например, с клавиатуры) пригоден оператор `>>`, но его применение ограничено, поскольку этот оператор считает пробелы разделителями. Допустим, в программе содержатся операторы:

```
char name[20];  
cout << "/ninput name: ";  
cin >> name;
```

Если в сеансе работы с программой в ответ на запрос

```
input name:
```

вести текст: **Петя Иванов**, переменной **name** будет присвоено только значение “Петя”, т.к. оператор `>>` считает пробел разделителем, который сигнализирует о завершении ввода значения.

Очевидно, использовать здесь посимвольный ввод с помощью рассмотренной выше функции **get** тоже неудобно.

Для ввода символьных строк часто более удобной оказывается функция **getline(...)**, имеющая 2 параметра:

```
<имя потока>. getline(<имя строковой переменной>, n);
```

Здесь **n** - длина строки без учета нуль-символа ‘\0’.

Например, оператор:

```
input.getline(str, 80);
```

позволяет считать из потока ввода **input** строку с пробелами длиной до 79 символов (последний, 80-й символ строки – служебный нуль-символ). Аналогичный оператор **cin.getline(str, 80);** позволяет получить такую же строку от пользователя с клавиатуры.

## Практическая работа 7

### Графы

**Граф** - это пара  $(V, E)$ , где  $V$  - конечное непустое множество вершин, а  $E$  - множество неупорядоченных пар  $(u, v)$  вершин из  $V$ , называемых ребрами.

Ребро  $s=(u, v)$  соединяет вершины  $u$  и  $v$ . Ребро  $s$  и вершина  $u$  (а также  $s$  и  $v$ ) называются **инцидентными**, а вершины  $u$  и  $v$  **смежными**. Степень вершины равна числу инцидентных ей ребер.

**Ориентированный граф**, или **орграф**,  $(V, E)$  отличается от обычного графа тем, что  $E$  - это множество упорядоченных пар  $(u, v)$  вершин, называемых дугами. Дуга  $(u, v)$  ведет от вершины  $u$  к вершине  $v$ . Вершина  $u$  называется **предшественником**  $v$ , а вершина  $v$  - **преемником**  $u$ . Графы представляются в программе чаще всего в виде **матрицы смежности** или **матрицы инцидентности**.

0 1 2 3 4 5 6	a b c d e
0 1 1 0 0 0 0	0 1 1 0 0 0
1 1 0 1 0 0 0	1 1 0 1 0 0
2 1 1 0 1 0 0	2 0 1 1 1 0
3 0 0 1 0 0 0	3 0 0 0 1 0
4 0 0 0 0 0 0	4 0 0 0 0 0
5 0 0 0 0 0 1	5 0 0 0 0 1
6 0 0 0 0 0 1 0	6 0 0 0 0 1

а) матрица смежности

б) матрица инцидентности

В матрице смежности 1 на пересечении  $i$ -й строки и  $j$ -го столбца означает, что вершины  $i$  и  $j$  смежны, а в матрице инцидентности – что вершина  $i$  и ребро  $j$  инцидентны. Для орграфа элемент матрицы смежности  $g[i][j]=1$ , если есть дуга  $i \rightarrow j$ .

Описание на Си графа, представленного в виде матрицы смежности:

```
int g[NMAX][NMAX];
```



где NMAX - это константа, задающая максимальное число вершин в графе.

Граф, представленный в виде матрицы инцидентности, можно описать так:

```
int g[NMAX][RMAX];
```

где RMAX - константа, задающая максимальное число ребер в графе (зависит от максимального числа вершин NMAX :  $RMAX = NMAX * (NMAX - 1) / 2$ , если граф без петель).

Здесь ребра пронумерованы, начиная с 0.

Внешнее представление графа может отличаться от внутреннего.

Например, граф можно задать в виде количества вершин и последовательности ребер, где каждое ребро - пара смежных вершин:

### Задача

Задан граф без петель в виде количества вершин  $n \leq 7$  и матрицы смежности. Сформировать для этого графа матрицу инцидентности.

### Программа

```
#include <stdio.h>
#include <conio.h>

/* Глобальные данные */
#define NMAX 7 /* максимальное число вершин графа */
#define RMAX 21 /* максимальное число ребер */
int g1 [NMAX][NMAX] , /* матрица смежности */
    g2 [NMAX][RMAX] , /* м-ца инцидентности */
    n , /* количество вершин */
    k ; /* количество ребер */
/* функция ввода матрицы смежности */
void VVOD_MATR_SM ()
/* Глобальные данные: g1,n */
{ int i,j ; /* параметры циклов */
    printf ("Введите матрицу смежности:\n\n");
```

```

printf (" | ");
for (j=0; j<n; j++) printf ("%d ",j);
putchar ('\n');
for (i=0; i<2*n+2; i++) putchar ('-');
for (i=0; i<n; i++)
    { printf ("\n%d| ",i);
      for (j=0; j<n; j++) scanf ("%d",&g1[i][j]);
    }
putchar ('\n');
}
/* функция вывода матрицы инцидентности */
void VIVOD_MATR_IN ()
    /* Глобальные данные: g2,n,k */
    { int i,l; /* параметры циклов */
      printf ("Матрица инцидентности\n\n");
      printf (" | ");
      for (l=0; l<k; l++) printf ("%d ",l);
      putchar ('\n');
      for (i=0; i<2*k+2; i++) putchar ('-');
      for (i=0; i<n; i++)
          { printf ("\n%d| ",i);
            for (l=0; l<k; l++)
                if (l<10) printf ("%d ",g2[i][l]);
                else printf ("%d ",g2[i][l]);
          }
      putchar ('\n');
    }
/* главная функция */
void main()
    { int i,j,l; /* индексы элементов матриц g1,g2 */

```

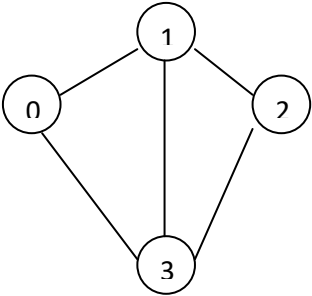
```

printf ("\nВведите количество вершин:");
scanf ("%d",&n);
VVOID_MATR_SM(); /* ввод матрицы смежности g1 */
    /* Формирование матрицы инц-ти g2 */
/* Обнулять м-цу g2 не нужно, т.к. глобальные и статические
    массивы автоматически инициализируются нулем ! */
k=0;
for (i=0; i<n; i++)
    for (j=i; j<n; j++)
        if (g1[i][j])
            { g2[i][k]=1;
              g2[j][k]=1;
              k++;
            }
VIVOD_MATR_IN(); /* вывод G2 */
getch();
}

```

### Тесты

1. Исходные данные:                      Ожидаемый результат:

n=4	матрица смежности	м-ца инцидентности
	<pre>   0 1 2 3 ----- 0  0 1 0 1 1  1 0 1 1 2  0 1 0 1 3  1 1 1 0 </pre>	<pre>   0 1 2 3 4 ----- 0  1 1 0 0 0 1  1 0 1 1 0 2  0 0 1 0 1 3  0 1 0 1 1 </pre>

2. Граф, изображенный на рис.1а.  $n=NMAX=7$ . Вид матрицы смежности см. на рис.2а. Ожидаемый результат: матрица инцидентности, приведенная на рис. 2б (только индексы столбцов цифровые).

3. Полный граф (все вершины смежны между собой), число вершин максимальное.

Исходные данные:

$n=7$ , матрица смежности:

```

| 0 1 2 3 4 5 6
-----
0| 0 1 1 1 1 1 1
1| 1 0 1 1 1 1 1
2| 1 1 0 1 1 1 1
3| 1 1 1 0 1 1 1
4| 1 1 1 1 0 1 1
5| 1 1 1 1 1 0 1
6| 1 1 1 1 1 1 0

```

Ожидаемый результат:

матрица инцидентности:

```

| 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
-----
0| 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1| 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
2| 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0
3| 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0
4| 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 0
5| 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1
6| 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 1 1

```

4. В графе нет ребер.

Исходные данные:

$n=3$  м-ца смежности

```

| 0 1 2
-----

```

Ожидаемый результат:

м-ца инцидентности

```

|
--

```

$0_1 0 0 0$	$0_1$
$1_1 0 0 0$	$1_1$
$2_1 0 0 0$	$2_1$

### Задания

1. Задан граф в виде количества вершин  $n \leq 10$  и последовательности ребер (каждое ребро задается парой смежных вершин). Получить матрицу смежности.

- Напечатать матрицу смежности. Проверить, есть ли в графе петли.
- Напечатать матрицу смежности. Проверить, есть ли в графе вершины, не смежные с другими.
- Напечатать для каждой вершины номера смежных вершин.
- Проверить, есть ли в графе вершина, смежная со всеми другими вершинами.
- Определить степень каждой вершины графа.
- Напечатать номера вершин со степенью 1.

2. Задан оргграф в виде количества вершин  $n \leq 10$  и последовательности дуг (дуга задается парой “предшественник преемник”).

- Напечатать номера вершин, имеющих более двух преемников.
- Напечатать номера вершин, не имеющих предшественников.
- Для каждой вершины напечатать номера всех предшественников.
- Проверить, есть ли в графе вершины, имеющие только одного преемника.

3. Задан оргграф в виде количества вершин  $n \leq 10$  и матрицы смежности.

- Напечатать номера вершин, имеющих и предшественников и преемников.
- Напечатать список дуг оргграфа.
- Напечатать номер вершины, имеющей наибольшее число преемников.

4. Задан граф без петель в виде количества вершин  $n \leq 7$ , количества ребер  $k \leq 21$  и матрицы инцидентности.

- а) Для каждой вершины напечатать список инцидентных ей ребер.
- б) Определить наибольшее число смежных между собой ребер, инцидентных одной и той же вершине.
- в) Проверить, есть ли вершины со степенью 0.
- г) Напечатать номера вершин, инцидентных только одному ребру.

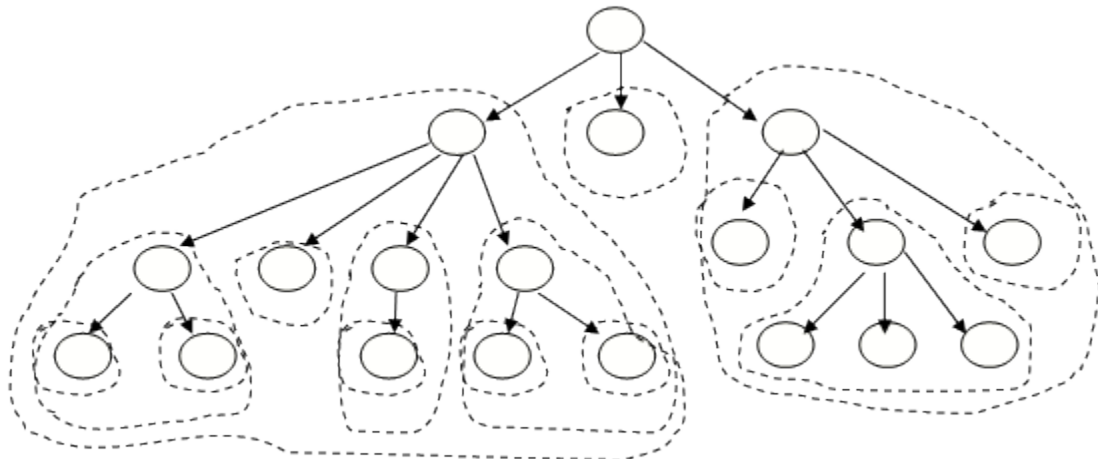
### Деревья

Структуры данных типа “дерево” широко используются в программной индустрии. В отличие от списковых структур деревья можно назвать **нелинейными** структурами. Любое дерево состоит из элементов – узлов или вершин, которые по определенным правилам связаны друг с другом. В списковых структурах за текущей вершиной (если она не последняя) всегда следует **только одна** вершина, тогда как в древовидных структурах таких вершин может быть **несколько**. Для дерева определена **единственная** начальная вершина (**корень** дерева) и **множество** конечных (терминальных) вершин (**листья**). От корня к листьям существует **множество** путей, тогда как в списках (линейных структурах!) путь всегда один.

Математически дерево рассматривается как частный случай графа, в котором отсутствуют замкнутые пути (циклы). Исторически сложилось так, что при описании деревьев используется «генеалогическая» терминология, основанная на понятиях **«родительская»** вершина (parent node) и **«дочерние»** вершины или вершины-**«потомки»** (childnode).

Разветвляющаяся структура дерева затрудняет его формальное определение и здесь на выручку приходит понятие **рекурсии**. Дерево является типичным примером **рекурсивно определённой структуры** данных, поскольку оно определяется в терминах самого себя. Как известно, рекурсия позволяет сводить **сложный** объект к набору более **простых** объектов той же природы. Дерево с N вершинами сводится к **набору** деревьев с **меньшим** числом вершин, каждое из них точно так же сводится к **своему** набору поддеревьев с еще меньшим числом вершин, до тех пор, пока не будут получены **простейшие** деревья с 1 или 0 вершин.

На рисунке ниже показано сведение дерева с 19 вершинами к трем поддеревьям с 10, 1 и 7 вершинами, каждое из которых сводится к своему набору своих поддеревьев. Например, самое большое поддерево с 10 вершинами сводится к четырем поддеревьям с 3, 1, 2 и 3 вершинами.



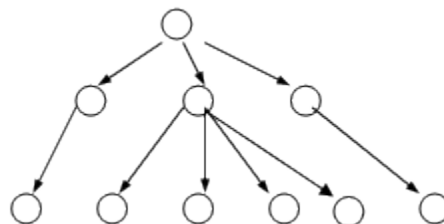
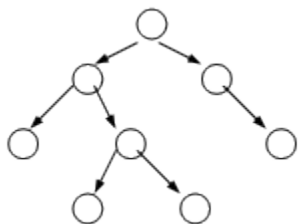
**Формальное рекурсивное определение** дерева с базовым типом  $T$  задается двумя **правилами**:

- это либо **пустое** дерево (не содержащее ни одного узла)
- либо некоторая **вершина** типа  $T$ , с которой связано **конечное** число деревьев с тем же базовым типом  $T$ , называемых **поддеревьями**

Существует много **разновидностей** деревьев, но, пожалуй, две самые **важные** разновидности возникают в зависимости от **числа** возможных **потомков** у каждой вершины:

Если у **КАЖДОЙ** вершины может быть **не более двух** потомков (т.е. два, один или ноль), то такие деревья называют **двоичными** (бинарными).

Если число потомков у вершины может быть **любым** (естественно – не бесконечным), то дерево называется **недвоичным** или **сильноветвящимся**.

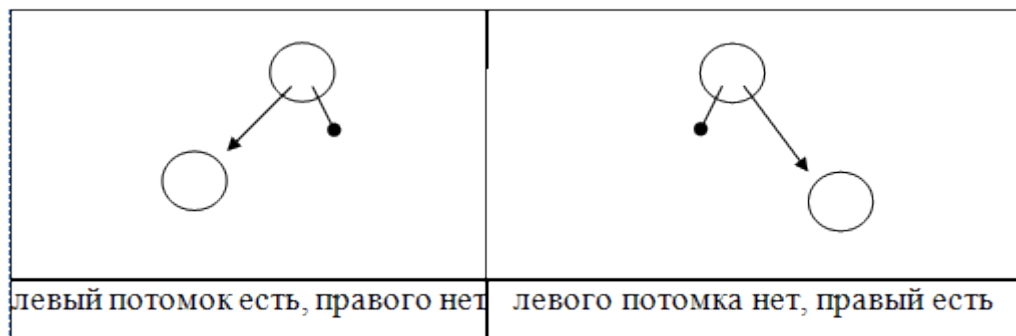


Двоичные деревья имеют большое самостоятельное значение, например – при эффективной организации поисковых операций. Кроме того,

двоичные деревья являются одним из способов представления недвоичных деревьев. Поэтому в дальнейшем сначала будут рассмотрены именно двоичные деревья, а затем – недвоичные.

Еще одной разновидностью деревьев являются **упорядоченные** деревья: для них важен **порядок** следования потомков. Для таких деревьев вводится понятие **левый** и **правый** потомок (для двоичных деревьев) или более левый/правый (для недвоичных деревьев).

Пример двух **разных** упорядоченных двоичных деревьев:



При использовании деревьев часто встречаются такие понятия как **путь** между начальной и конечной вершиной (последовательность проходимых ребер или вершин), **высота** дерева (наиболее длинный путь от корневой вершины к терминальным).

Далее рассмотрим общие вопросы, связанные с реализацией **двоичных упорядоченных** деревьев. Каждая вершина такого дерева должна иметь следующие поля:

-одно или несколько **информационных** полей, содержащих обрабатываемые данные (если эти данные достаточно большие по объему, можно рассмотреть вопрос об их **отдельном** от вершины хранении с адресацией соответствующими указателями)

-**связующее** поле для указания возможного **левого** потомка

-**связующее** поле для указания возможного **правого** потомка

Отсюда следует, что вершина дерева описывается как **запись/структура** с набором необходимых полей. Все вершины одного

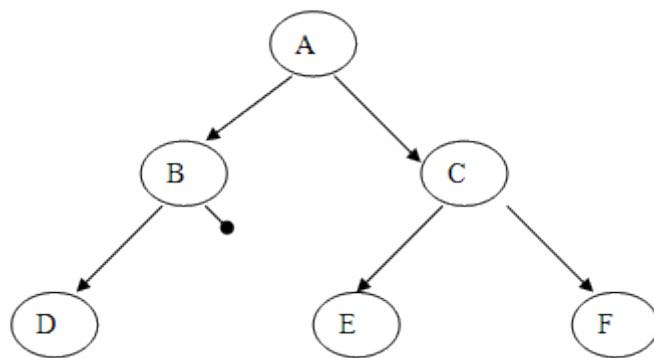


дерева имеют одну и ту же структуру (однотипны). Двоичное дерево, аналогично линейным структурам, можно реализовать **двумя** способами:

- на основе **массива записей** (статическая или непрерывная реализация)
- на базе механизма динамического распределения памяти и указательных переменных (динамическая реализация)

В первом случае записи-вершины являются элементами массива, **индексируются** порядковыми номерами и связующие поля у каждой вершины должны содержать **номера** ячеек, где находятся левый и правый потомок этой вершины. Если потомка нет, соответствующее связующее поле должно содержать некоторый **фиктивный** индекс (например 0 при индексации элементов в массиве с 1, или -1 при индексации с 0). Для использования такой структуры надо знать индекс размещения в массиве корневой вершины. По умолчанию самое простое – разместить корень в первой ячейке массива.

**Пример.** Дано небольшое двоичное дерево с символьными элементами:



Тогда его размещение в массиве может выглядеть так:

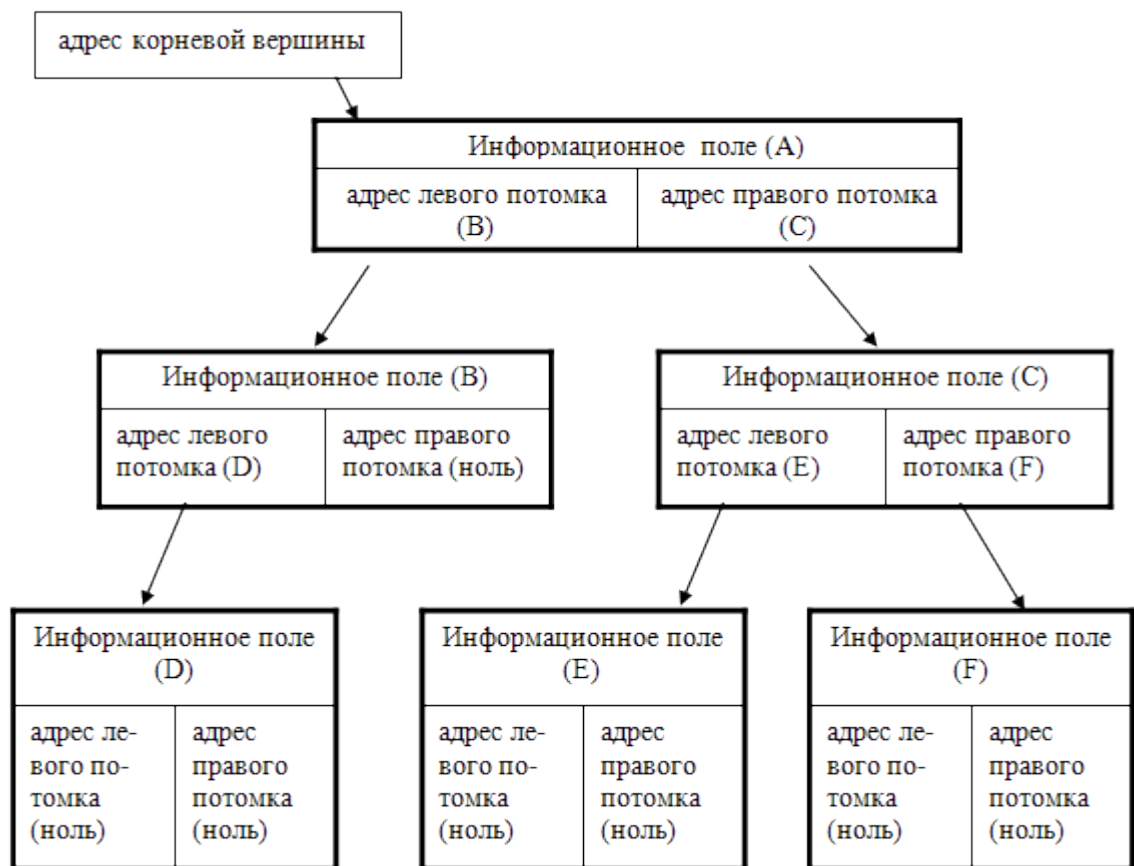
Индекс	1	2	3	4	5	6	7	8	9
Информационное поле	A		C	B	F	E		D	
Индекс левого потомка	4		6	8	0	0		0	
Индекс правого потомка	3		5	0	0	0		0	

Однако чаще двоичные деревья реализуются **динамически**. В этом случае его вершины размещаются в динамически распределяемой памяти и связующие поля содержат **адреса** потомков. Если потомка нет – адресное

поле содержит **пустое** (нулевое) значение. Отсюда следует, что у терминальных вершин оба адресных поля нулевые. Это, кстати, один из недостатков динамической реализации: примерно половина вершин в двоичном дереве являются терминальными и у всех них адресные поля нулевые.

Для доступа к дереву достаточно знать адрес размещения в памяти корневой вершины (аналогично адресу первого элемента списка).

**Например**, рассмотренное выше дерево с шестью вершинами в динамической реализации можно условно показать так:



Для динамической реализации необходимы **стандартные** объявления:

- описание **структуры** вершин дерева с двумя **обязательными** указательными полями (в Паскале для этого необходим указательный тип)
- описание одной **основной** указательной переменной для адресации **корневой** вершины

Паскаль	Си
<code>type pTreeNode = ^TTreeNode;</code>	<code>struct TreeNode</code>

<pre> TTreeNode = record info : char; left : pTreeNode; right : pTreeNode; end; var pRoot : pTreeNode; </pre>	<pre> { char info; struct TreeNode *left; struct TreeNode *right; }; struct TreeNode *pRoot; </pre>
---	---

Тогда пустое дерево просто определяется установкой переменной pRoot в нулевое значение (например – **nil**).

### **Основные операции с двоичными деревьями:**

- обход всех вершин дерева в некотором порядке (общая операция)
- добавление новой вершины как потомка заданной вершины
- удаление заданной вершины
- поиск заданной вершины

Последние три операции имеют разную реализацию в зависимости от типа дерева (обычное двоичное дерево или специальное поисковое дерево).

## Список литературы

1. Немцова, Т. И. Программирование на языке высокого уровня. Программирование на языке С++ : учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев ; под ред. Л.Г. Гагариной. — Москва : ФОРУМ : ИНФРА-М, 2023. — 512 с. + Доп. материалы [Электронный ресурс]. — (Среднее профессиональное образование). - ISBN 978-5-8199-0699-6. - Текст : электронный. - URL: <https://znanium.com/catalog/product/1916204> (дата обращения: 28.06.2023). — Режим доступа: по подписке.

2. Рейзлин, В. И. Язык С++ и программирование на нём : учебное пособие / В. И. Рейзлин. — 3-е изд., перераб. — Томск : ТПУ, 2021. — 206 с. — ISBN 978-5-4387-0975-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/246239> (дата обращения: 28.06.2023). — Режим доступа: для авториз. пользователей.

3. Ефимова Ю.В. Программирование на языке высокого уровня: Практикум. - Казань: Изд-во Казан. гос. техн. ун-та, 2012. - 32 с.

4. Чукич, И. Функциональное программирование на С++ : учебное пособие / И. Чукич ; перевод с английского В. Ю. Винника, А. Н. Киселева. — Москва : ДМК Пресс, 2020. — 360 с. — ISBN 978-5-97060-781-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/140597> (дата обращения: 28.06.2023). — Режим доступа: для авториз. пользователей.

5. Программирование на языке Си/А.В.Кузин, Е.В.Чумакова - М.: Форум, НИЦ ИНФРА-М, 2015. - 144 с. - Режим доступа: <http://znanium.com/bookread2.php?book=505194>

6. Язык Си: кратко и ясно: Учебное пособие / Д.В. Парфенов. - М.: Альфа-М: НИЦ ИНФРА-М, 2014. - 320 с. - Режим доступа: <https://znanium.com/read?id=356040>

7. Программирование графики на С++. Теория и примеры : учеб. пособие / В.И. Корнеев, Л.Г. Гагарина, М.В. Корнеева. — М. : ИД «ФОРУМ»

: ИНФРА-М, 2017. — 517 с. - Режим доступа:  
<http://znanium.com/bookread2.php?book=562914>

8. Программирование на языке C++: Учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: ИНФРА-М, 2012. - 512 с. - Режим доступа:  
<http://znanium.com/bookread2.php?book=244875>