Министерство науки и высшего образования Российской Федерации

Южно-Российский государственный
политехнический университет (НПИ) имени М.И. Платова

Д.Н. Кущий, А.А. Масленников

ИНФОРМАТИКА

**Методические указания
к лабораторным работам и самостоятельной работе
студентов бакалавриата направлений подготовки
«Математическое обеспечение и администрирование
информационных систем»,
«Информатика и вычислительная техника»,
«Программная инженерия» и «Прикладная математика»**

Новочеркасск
ЮРГПУ(НПИ)
2022

УДК 681.3 (076.5)
ББК 32.973.233я 73

Рецензент – кандидат технических наук, доцент,
доцент кафедры «Программное обеспечение вычислительной техники»
Панфилов Александр Николаевич

Кущий Д.Н., Масленников А.А.

Информатика: методические указания к лабораторным работам и самостоятельной работе студентов бакалавриата направлений подготовки «Математическое обеспечение и администрирование информационных систем», «Информатика и вычислительная техника», «Программная инженерия» и «Прикладная математика» / Д.Н. Кущий, А.А. Масленников; Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2022. – 32 с.

Методические указания содержат описание лабораторных занятий, краткую теорию с примерами и варианты заданий. Приведены требования к содержанию и оформлению отчета по лабораторным работам, список тем и вопросов для самостоятельного изучения, а также экзаменационные вопросы и типовые задачи.

УДК 681.3 (076.5)
ББК 32.973.233я 73

1. ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа 1

СОЗДАНИЕ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ, УПРАВЛЕНИЕ ВВОДОМ И ВЫВОДОМ СРЕДСТВАМИ ЯЗЫКА *C*

Цель: изучение основных типов данных, способов описания переменных различных типов, операторов языка и организации ввода-вывода.

Краткая теория

В языке *C* нет встроенных средств ввода и вывода – они осуществляются с помощью функций, типов и объектов, которые находятся в стандартных библиотеках.

Для ввода/вывода данных в C используются функции, которые описываются в библиотечном файле *stdio.h*.

Синтаксис функции вывода информации на консоль:

```
printf (форматная строка, список аргументов);
```

где `форматная строка` – строка символов, заключенных в кавычки, которая показывает, как должны быть напечатаны аргументы, `список аргументов` – переменные, значения которых следует напечатать.

Например:

```
printf (”Значение числа Пи равно %f\n”, pi);
```

Форматная строка может содержать: символы, печатаемые в текстовом виде, спецификации преобразования, управляющие символы. Более подробная информация представлена в соответствующей лекции.

Синтаксис функции считывания информации с консоли:

```
scanf(форматная строка, список аргументов);
```

В качестве аргументов используются адреса переменных.

Например: `scanf(“%d %f”, &x, &y);`

Пример. Написать программу для вычисления выражения, определяемого соотношением. Блок-схема приведена на рис.1.1.

Листинг программы с комментариями:

```
// Подключение библиотеки стандартного ввода-вывода
#include <stdio.h>
```

```
// подключение библиотеки контроля процесса выпол-
нения
#include <stdlib.h>
// подключение математической библиотеки
#include <math.h>
// для языковых настроек
#include <locale.h>
int main()
{
    // подключение русского языка
    setlocale(LC_ALL, "");
    // объявление переменных
    float z, x, y;
    // приглашение к вводу переменной x
    printf("Введите x: ");
    // считывание значения переменной x
    scanf("%f", &x);
    // приглашение к вводу переменной y
    printf("Введите y: ");
    // считывание значения переменной y
    scanf("%f", &y);
    //проверка корректности значений
    if ((x + pow(fabs(y), (1. / 4.)))>0)
    {
        // основное вычисление
        z = pow(2, -x)*
           sqrt(x + pow(fabs(y), (1. / 4.)));
       // вывод результата на экран
        printf("z = %f\n", z);
    }
    else
        printf("Значение выражения не может быть
вычислено\n", z);
    system("PAUSE"); // задержка консоли
    return 0;
}
```

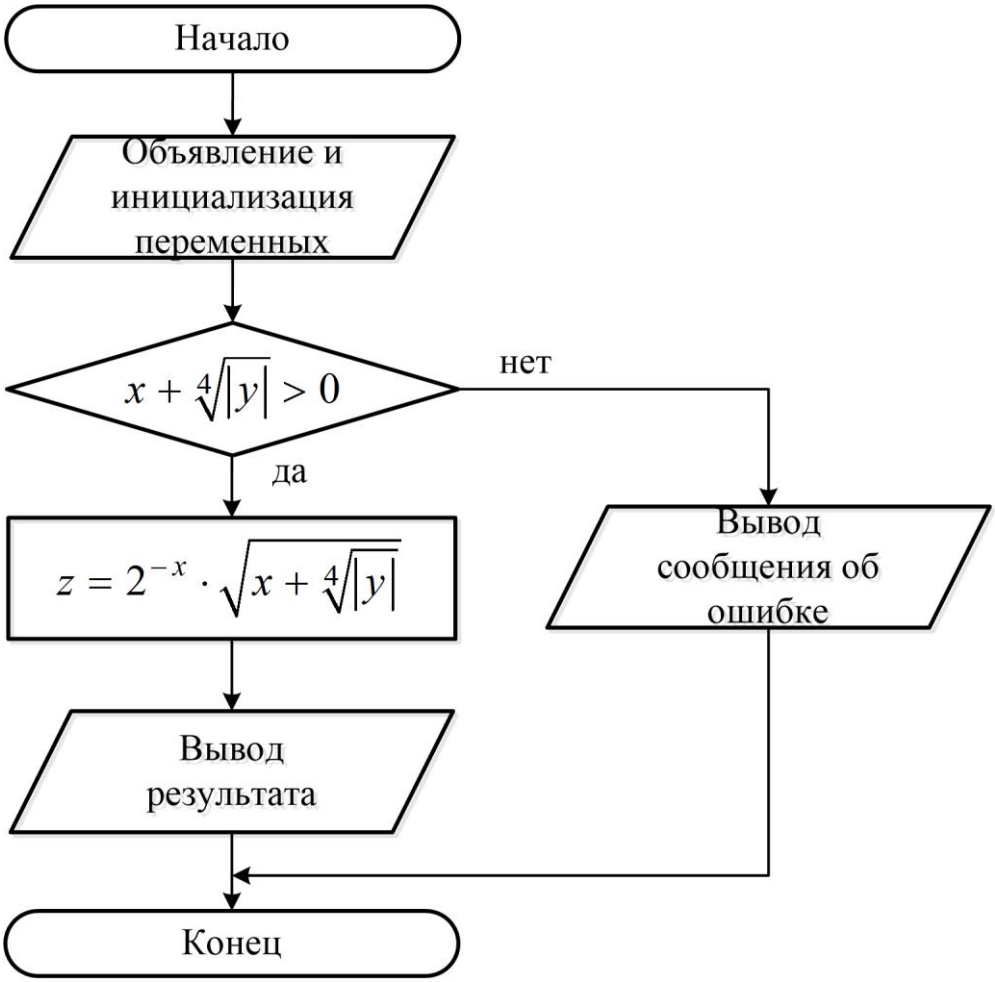

Рис. 1.1. Блок-схема алгоритма

Рассмотренная реализация при вводе некорректных значений *x* и *y*, завершит свою работу. Для возможности повторного ввода этих значений без перезапуска исполняемого модуля необходимо использовать оператор повтора (цикла) и проверять корректность данных сразу:

```
do{
    // приглашение к вводу переменной x
    printf("Enter x: ");
    // считывание значения переменной x
    scanf("%f", &x);
    // приглашение к вводу переменной y
    printf("Enter y: ");
    // считывание значения переменной y
    scanf("%f", &y);
} while((x + pow(fabs(y), (1. / 4.)))<0);
```

Варианты заданий

В программе должно быть реализовано меню, содержащее возможность выхода из программы. Ввод-вывод данных необходимо реализовать с использованием *scanf, printf*. После выполнения задачи должна быть реализована возможность возврата в меню.

1. Написать программу для вычисления периметра и площади трапеции, а также площади описанной окружности. Основания и высота трапеции задаются пользователем.
2. Написать программу для вычисления углов равнобедренного треугольника и длины его боковой стороны по известным высоте и основанию. Высота и основание равнобедренного треугольника задаются пользователем.
3. Написать программу для вычисления диаметра и длины окружности, а также площади вписанного в нее правильного треугольника. Радиус окружности задается пользователем.
4. Написать программу для вычисления периметра и площади прямоугольника, а также длины описанной окружности. Стороны прямоугольника задаются пользователем.
5. Написать программу, которая переводит время из минут и секунд в секунды. Например, 2ч 51 минут переводит в 60*2+51=171 секунда.
6. Написать программу, которая выводит таблицу квадратов следующих за введенным числом пяти положительных нечетных чисел.
7. Написать программу для вычисления дня недели по введенной дате для григорианского календаря.
8. Написать программу, которая проверяет, является ли введенное пользователем целое число простым (простым называется число, которое делится только на единицу и на само себя).
9. Написать программу вычисления площади боковой поверхности куба и площади поверхности сферы, описанной около него. Длина ребра куба задается пользователем.
10. Написать программу определения типа треугольника (равносторонний, равнобедренный, разносторонний) по трем числам, являющимся длинами сторон треугольника. Длины сторон задаются пользователем.
11. Написать программу нахождения среднего арифметического кубов и среднего геометрического модулей двух чисел. Числа задаются пользователем.

12. Написать программу для проверки того, что четыре точки A, B, C, D на плоскости образуют квадрат ABCD. Координаты точек задаются пользователем.

13. Написать программу для определения количества точек пересечения двух окружностей по координатам центров и радиусам. Координаты центров и радиусы окружностей задаются пользователем.

14. Написать программу для вычисления количества високосных лет между двумя годами. Начало и конец периода вводит пользователь.

15. Написать программу, которая после введенного с клавиатуры числа (в диапазоне от 1 до 99), обозначающего денежную единицу, дописывает слово «копейка» в правильной форме.

16. Написать программу, которая после введенного с клавиатуры числа (в диапазоне от 1 до 999), обозначающего денежную единицу, дописывает слово «рубль» в правильной форме.

17. Написать программу, которая проверяет, является ли введенное пользователем целое число отрицательным, четным и кратным 5 одновременно.

18. Написать программу вычисления сопротивления электрической цепи, состоящей из 2-х сопротивлений. Сопротивления могут быть соединены последовательно (r_1+r_2) или параллельно (r_1*r_2/(r_1+r_2)).

19. Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 3% предоставляется, если сумма покупки больше n рублей, в 5% − если сумма больше m рублей. Значения n и m вводятся пользователем.

20. Написать программу, которая проверяет, является ли год високосным. Использовать полное условие проверки.

Лабораторная работа 2

ВВОД И ВЫВОД В ФАЙЛОВЫЕ СТРУКТУРЫ СРЕДСТВАМИ ЯЗЫКА *C*

Цель: изучение принципов ввода-вывода с использованием файлов.

Краткая теория

Для удобства обращения информация в запоминающих устройствах хранится в виде файлов.

Файл – именованная область внешней памяти, выделенная для хранения массива данных. Данные, содержащиеся в файлах, имеют

самый разнообразный характер: программы на алгоритмическом или машинном языке; исходные данные для работы программ или результаты выполнения программ; произвольные тексты; графические изображения и т. п.

Каталог (***папка, директория***) – именованная совокупность байтов на носителе информации, содержащая название подкаталогов и файлов, используется в файловой системе для упрощения организации файлов.

Файловой системой называется функциональная часть операционной системы, обеспечивающая выполнение операций над файлами. Примерами файловых систем являются *FAT* (*FAT – File Allocation Table*, таблица размещения файлов), *NTFS, UDF* (используется на компакт-дисках).

Для программиста открытый файл представляется как последовательность считываемых или записываемых данных. При открытии файла с ним связывается *поток ввода-вывода*. Выводимая информация записывается в поток, вводимая информация считывается из потока.

Когда поток открывается для ввода-вывода, он связывается со стандартной структурой типа *FILE*, которая определена в *stdio.h*. Структура *FILE* содержит необходимую информацию о файле.

Открытие файла осуществляется с помощью функции *fopen*(), которая возвращает указатель на структуру типа *FILE*, который можно использовать для последующих операций с файлом.

Синтаксис функции:

```
FILE *fopen(name, type);
```

где *name* – имя открываемого файла (включая путь), *type* – указатель на строку символов, определяющих способ доступа к файлу:

- "*r*" – открыть файл для чтения (файл должен существовать);
- "*w*" – открыть пустой файл для записи; если файл существует, то его содержимое теряется;
- "*a*" – открыть файл для записи в конец (для добавления); файл создается, если он не существует;
- "*r+*" – открыть файл для чтения и записи (файл должен существовать);
- "*w+*" – открыть пустой файл для чтения и записи; если файл существует, то его содержимое теряется;

• "*a+*" – открыть файл для чтения и дополнения, если файл не существует, то он создаётся.

Возвращаемое значение – указатель на открытый поток. Если обнаружена ошибка, то возвращается значение *NULL*.

Функция *fclose*() закрывает поток или потоки, связанные с открытыми при помощи функции *fopen*() файлами. Закрываемый поток определяется аргументом функции *fclose*().

Возвращаемое значение: значение 0, если поток успешно закрыт; константа EOF, если произошла ошибка.

Пример. Проверить существование файла.

```
#include <stdio.h>
int main() {
  FILE *fp;
  char name[] = "my.txt";
  if ((fp = fopen(name, "r")) == NULL)
  {
    printf("Не удалось открыть файл");
    getchar();
    return 0;
  }
  // открыть файл удалось
  // требуемые действия над данными
  fclose(fp);
  return 0;
}
```

Для чтения символа из файла используется функция *fgetc*(), синтаксис которой имеет вид:

```
char fgetc(поток);
```

Аргументом функции является указатель на поток типа FILE. Функция возвращает код считанного символа. Если достигнут конец файла или возникла ошибка, возвращается константа EOF.

Для записи символа в файл используется функция *fputc* (), синтаксис которой имеет вид:

```
fputc(символ,поток);
```

Аргументами функции являются символ и указатель на поток типа FILE. Функция возвращает код считанного символа.

Функции *fscanf*() и *fprintf*() аналогичны функциям работы с консолью *scanf*() и *printf*(), но работают с файлами данных, и имеют первый аргумент – указатель на файл, их синтаксис имеет вид:

```
fscanf(поток, "ФорматВвода", аргументы);
fprintf(поток, "ФорматВывода", аргументы);
```

Функции *fgets*() и *fputs*() предназначены для ввода-вывода строк, они являются аналогами функций *gets*() и *puts*() для работы с файлами.

```
fgets(УказательНаСтроку,КоличествоСимволов,поток);
```

Символы читаются из потока до тех пор, пока не будет прочитан символ новой строки ‘*n*’, который включается в строку, или пока не наступит конец потока *EOF* или не будет прочитано максимальное количество символов. Результат помещается в указатель на строку и заканчивается нуль-символом ‘\0’. Функция возвращает адрес строки.

```
fputs(УказательНаСтроку,поток);
```

Копирует строку в поток с текущей позиции. Завершающий нуль-символ не копируется.

Пример. Ввести число и сохранить его в файле s1.txt. Считать число из файла s1.txt, увеличить его на три и сохранить в файле s2.txt.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  FILE *S1, *S2;
  int x, y;
  system("chcp 1251");
  system("cls");
  printf("Введите число : ");
  scanf("%d", &x);
  S1 = fopen("S1.txt", "w");
  fprintf(S1, "%d", x);
  fclose(S1);
  S1 = fopen("S1.txt", "r");
  S2 = fopen("S2.txt", "w");
  fscanf(S1, "%d", &y);
  y += 3;
  fclose(S1);
```

```
  fprintf(S2, "%d\n", y);
  fclose(S2);
  return 0;
}
```

Результат выполнения – два файла (рис. 1.2).

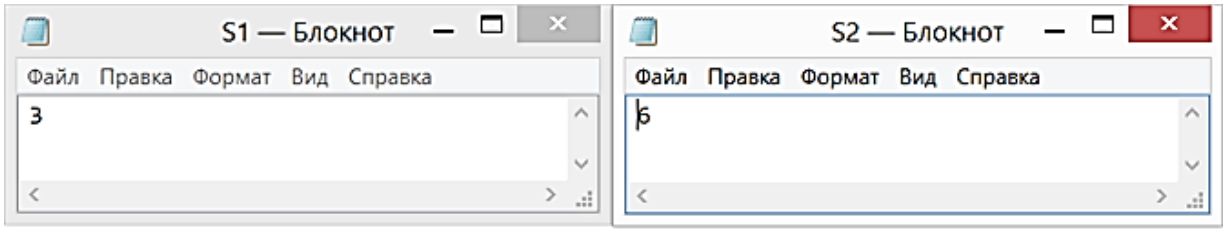

Рис. 1.2. Результаты работы программы

Варианты заданий

Задания аналогичны вариантам лабораторной работы 1. В написанные ранее программы следует внести следующие дополнения: исходные значения необходимых для расчетов переменных считываются из одного файла, а результат выполнения программы записывается в другой.

Лабораторная работа 3

РАБОТА С МАССИВАМИ И УКАЗАТЕЛЯМИ В ЯЗЫКЕ *C*

Цель: приобретение практических навыков в составлении программ с массивами и указателями.

Краткая теория

Массивы – структурированный тип данных с элементами одного и того же типа, имеющий одно имя и определенное количество элементов. Количество элементов определяет размер массива. Порядковый номер элемента массива называется его индексом. Число индексов называется размерностью массива, например, массив с двумя индексами называется двумерным массивом. Строка символов является массивом символов, вектор – массив чисел, матрица – массив векторов.

Обработка массивов выполняется следующим образом: объявление, ввод или инициализация элементов массива, преобразование и вывод.

Чтобы использовать массив, надо его объявить – выделить место в памяти компьютера, объём которой зависит от количества элементов и типа массива. Тип массива – это тип входящих в него элементов. Массивы могут быть разных типов: *int*, *float*, *char*, и т.д.

Массив объявляют так же, как и обычные переменные, но после имени массива в квадратных скобках записывается его размер:

```
// 2 массива по 10 и 20 целых чисел
int A[10], B[20];
// массив из 12 вещественных чисел
float C[12];
```

При объявлении массива можно сразу заполнить его начальными значениями, перечисляя их внутри фигурных скобок:

```
int A[4] = {2, 3, 12, 76};
```

Если в списке в фигурных скобках записано меньше чисел, чем элементов в массиве, то оставшиеся элементы заполняются нулями. Если чисел больше, чем надо, компилятор сообщает об ошибке. Например:

```
int A[4] = {2};
// последние три элемента равны 0
```

Для повышения универсальности программы размер массива лучше определять через константу. В этом случае для переделки программы для массива другого размера надо только поменять значение этой константы:

```
const int N = 20; //N - константа
main()
{   // размер массива задан через константу
    int A[N];
    //основные операторы программы
}
```

Каждый элемент массива имеет свой порядковый номер. Чтобы обратиться к элементу массива, надо написать имя массива и затем в квадратных скобках номер нужного элемента. Важно запомнить правило: элементы массивов в языке *C* нумеруются с нуля. Поэтому индекс последнего элемента массива на 1 меньше числа элементов в данном массиве.

Таким образом, если в массиве 10 элементов, он содержит элементы:

```
A[0], A[1], A[2], ..., A[9]
```

Номер элемента массива также называется его *индексом*. Вот примеры обращения к массиву *A*:

```
// прочитать значения A[3] и A[1]
x = (A[3] + 5)*A[1];
// записать новое значение в A[0]
A[0] = x + 6;
```

В языке *C* не контролируется *выход за границы массива*, то есть формально вы можете записать что-то в элемент с несуществующим индексом, например в *A*[345] или в *A*[-12]. Однако при этом вы стираете какую-то ячейку в памяти, не относящуюся к массиву, поэтому последствия такого шага непредсказуемы и во многих случаях программа «зависает».

Существует много способов ввода в зависимости от вашей задачи:

- элементы массива вводятся с клавиатуры вручную;
- массив заполняется случайными числами (например, для моделирования случайных процессов);
- элементы массива читаются из файла;
- массив заполняется в процессе вычислений.

Чтобы ввести массив в память, надо каждый его элемент обработать отдельно (например, вызвав для него функцию ввода *scanf*()). Для ввода и вывода массива обычно используется цикл *for*.

Для всех переменных выделяются участки памяти размером, соответствующим типу переменной. Программист имеет возможность работать непосредственно с адресами, для чего определен соответствующий тип данных – *указатель*. Указатель имеет следующий формат:

```
тип *имя указателя;
```

Например:

```
int *a; double *b, *d; char *c;
```

Знак «*» относится к имени указателя. Значение указателя соответствует первому байту участка памяти, на который он ссылается. На один и тот же участок памяти может ссылаться любое число указателей.

В языке *C* существует три вида указателей:

1. Указатель на объект известного типа. Содержит адрес объекта определенного типа.

2. Указатель типа *void*. Применяется, если тип объекта заранее не определен.

3. Указатель на функцию.

Над указателями можно провести две унитарные операции:

1. & (*взять адрес*). Указатель получает адрес переменной. Данная операция применима к переменным, под которые выделен соответствующий участок памяти.

2. * (*операция разыменования*). Предназначена для доступа к величине, расположенной по данному адресу.

Над указателями можно выполнять арифметические операции сложения, инкремента, вычитания, декремента и операции сравнения. При выполнении арифметических операций с указателями автоматически учитывается размер данных, на которые он указывает.

Указатели, как правило, используются при работе с динамической памятью (*heap*, или «куча»). Для работы с динамической памятью в языке *C* определены следующие функции: *malloc, calloc, realloc* и *free*.

В тех случаях, когда размер массива невозможно знать заранее (например, эта величина вводится человеком), можно использовать динамические массивы.

Функция *calloc* возвращает нетипизированный указатель на начало массива:

```
void* calloc(size_t n, size_t r);
```

где *n* – количество элементов массива; *r* – размер элемента в байтах.

Пример. Динамическое выделение памяти массиву с помощью *calloc*:

```
z = (int*)calloc(n, sizeof(int));
```

Функция *malloc* также возвращает нетипизированный указатель:

```
void*malloc(size_t r);
```

где *r* – размер элемента в байтах.

Данная функция выделяет блок памяти размером *r* байт в свободной памяти.

Пример. Динамическое выделение памяти массиву malloc:

```
z = (int*)malloc(n* sizeof(int));
```

В программировании встречается неприятная ситуация, которая называется «утечка памяти». Эта ситуация может возникнуть при использовании динамического выделения памяти. Для устранения этой ситуации рекомендуется освобождать неиспользуемую память с помощью функции `void free(void *p)`, где *p* – нетипизированный указатель, получивший значение в результате вызова *calloc* или *malloc*.

Пример. Заполнить массив равномерно распределенными случайными числами в интервале [*a*, *b*]. Реализовать вариант генерации целых и вещественных чисел.

```
#include <stdio.h>
#include <stdlib.h>  // rand(), srand()
#include <time.h>    // time()
#include <conio.h>

int intRandom(int a, int b)
{
    return  rand()%(b-a+1) + a;
}
float floatRandom(int a, int b)
{
    return  (float)rand()*(b-a)/RAND_MAX + a;
}
int* dynamicMemoryInt(int n)
{
    int *z = (int*)malloc(n * sizeof(int));
    return z;
}
float* dynamicMemoryFloat(int n)
{
    float *z = (float*)malloc(n * sizeof(float));
    return z;
}
int main()
{
    srand(time(NULL));
    int i, n,  a = -5, b = 10;
    int *X;
```

```
    float *Y;
    printf("Введите количество элементов в массиве
->");
    scanf("%d", &n);
    X=dynamicMemoryInt(n);
    Y=dynamicMemoryFloat(n);
    for (i = 0; i < n; i++)
        X[i] = intRandom(a,b);
    for (i = 0; i < n; i++)
       Y[i] = floatRandom(a,b);
    printf("\nInteger\n");
    for (i = 0; i < n; i++)
        printf("%4d",X[i]);
    printf("\nFloat\n");
    for (i = 0; i < n; i++)
        printf("%6.2f",Y[i]);
    return 0;
}
```

Примечание. В приведенном ниже примере массив *A* заполняется случайными целыми числами в интервале [-5,10], а массив *X* – случайными вещественными числами в том же интервале.

Для получения случайных чисел с равномерным распределением в интервале [*a*, *b*] надо использовать формулу:

```
k = rand() ( b − a + 1 ) + a;
```

Для вещественных чисел формула несколько другая:

```
x = rand()*(b - a)/RAND_MAX + a;
```

Здесь константа `RAND_MAX` – это максимальное случайное число, которое выдает стандартная функция *rand*().

Пример. Дан массив из *n* целых чисел. Найти наибольший элемент в массиве и его порядковый номер.

```
    for (i=0; i<n; i++)
   {
        printf("\n Input the Array Element->  ");
        scanf ("%d", &a[i]);
    };
    for (i=1,max=a[0],nom=0; i<n; i++)
```

```
        if (max<a[i])
       {
           nom=i;
           max=a[i];
        }
        printf("\n Data Array : \n");
        for (i=0; i<n; i++)printf ("%6d", a[i]);
           printf  ("\nМаксимальный  элемент  %4d,
его индекс %4d  " , max, nom+1);
```

Пример. Удалить из одномерного массива все отрицательные элементы

```
        for (i=0; i<n; i++)
           if (a[i]<0)
           {
               for (j=i+1; j<n; j++) a[j-1]=a[j];
                   n--; i--;
           }
```

Варианты заданий

1. Задан массив из *k* действительных чисел. Заменить все его элементы, большие заданного *Z*, этим числом. Подсчитать количество замен.
2. Задан массив из *k* целых положительных чисел. Найти среди них те, которые являются квадратами некоторого числа *m*.
3. Задан массив из *k* целых чисел. Найти наиболее часто встречающееся число. Если таких чисел несколько, то определить наименьшее из них.
4. Задан массив из *k* целых чисел. Найти сумму минимального и максимального элемента и заменить на нее значение последнего элемента.
5. Задан массив из *k* символов. Определить количество различных элементов в массиве.
6. Задан массив из *k* целых чисел. Все элементы, равные нулю, поставить сразу после максимального элемента данного массива.

7. Заданы два одномерных массива A и B с одинаковым количеством элементов. Составить программу подсчёта суммы элементов с нечётными индексами в массиве B и произведения отрицательных элементов в массиве A.

8. Задан массив из k чисел. Заменить все отрицательные элементы, стоящие перед минимальным элементом массива, квадратом первого элемента массива.

9. Задан массив из k чисел. Заменить единицами все положительные элементы, имеющие четный порядковый номер и идущие после минимального элемента массива.

10. Задан массив из k целых чисел. Заменить нулями все элементы, меньшие, чем элемент массива, расположенный слева от максимального.

11. Задан массив из k символов. Преобразовать массив следующим образом: сначала должны стоять цифры, входящие в массив, а затем все символы.

12. Задан массив из k символов латинского алфавита. Вывести на экран в алфавитном порядке все символы, которые входят в этот массив по одному разу.

13. Задан массив из k чисел. Отрицательные элементы, имеющие четный порядковый номер, переписать в начало массива.

14. Задан массив из k чисел. Все положительные элементы, расположенные между отрицательными, поставить после минимального элемента массива.

15. Задан массив из k чисел. Заменить все отрицательные элементы, расположенные между положительными, последним элементом массива.

16. Задан массив из k чисел. Все положительные элементы, имеющие нечетный порядковый номер, переписать в конец массива.

17. Задан массив из k целых чисел. Определить сколько среди них чисел больших некоторого заданного Z, и сколько чисел меньше него.

18. Задан массив из k символов. Заменить в нем на «*» повторные вхождения каждого символа.

19. Задан массив из k чисел. Найти числа, входящие в массив только один раз.

20. Задан массив из k целых чисел. Найти значение наименьшего среди тех элементов массива, что лежат вне интервала $[a, b]$.

Лабораторная работа 4

РАБОТА СО СТРУКТУРАМИ НА ЯЗЫКЕ *C*, СОЗДАНИЕ И ЗАПОЛНЕНИЕ

Цель: приобретение практических навыков в составлении алгоритмов и программ со структурами.

Краткая теория

Структура – это составной тип данных, в котором под одним именем объединены данные различных типов. Отдельные данные структуры называются *полями*. Объявление структуры осуществляется с помощью ключевого слова *struct*, за которым идет ее имя и далее список элементов, заключенных в фигурные скобки:

```
struct имя
{
    тип_элемента_1 имя_элемента_1;
    тип_элемента_2 имя_элемента_2;
    тип_элемента_n имя_элемента_n;
};
```

Правила работы с полями структуры идентичны работе с переменными соответствующих типов. К полям структуры можно обращаться через составное имя. Формат обращения:

имя_структуры.*имя_поля*

или

указатель_на_структуру–>*имя_поля*

Варианты заданий

Программа, демонстрирующая работу с созданной структурой, должна состоять из нескольких функций, в коде должен использоваться *typedef*. Память для размещения массива структур следует выделять динамически, а также необходимо реализовать возможность заполнения структуры из файла и ввода данных с клавиатуры.

1. Дана информация о 10 людях. Структура имеет следующие поля: фамилия, имя, отчество, пол, дата рождения (год, месяц, число), номер телефона, домашний адрес (город, улица, дом, квартира). Найти средний возраст.

2. Дана информация о 10 студентах. Структура имеет следующие поля: фамилия, имя, отчество, факультет, курс, группа, дата рождения

(год, месяц, число), номер телефона, домашний адрес (почтовый индекс, город, улица, дом, квартира). Вывести списки студентов по факультетам.

3. Дана информация о 10 покупателях. Структура имеет следующие поля: фамилия, имя, отчество, дата рождения (год, месяц, число), телефон, *e-mail,* название магазина, скидка по дисконтной карте. Вывести список покупателей по убыванию размера скидки по карте.

4. Дана информация о 10 пациентах. Запись имеет вид: фамилия, имя, отчество, пол дата рождения (год, месяц, число), мобильный телефон, домашний телефон, домашний адрес (индекс, улица, дом, квартира), давление. Вывести данные о самом молодом пациенте с повышенным давлением (больше 140).

5. Дана информация о 10 владельцах автомобиля. Структура имеет следующие поля: фамилия, имя, отчество, дата рождения (индекс, улица, дом), номер телефона, марка автомобиля, номер автомобиля, номер техпаспорта. Вывести информацию о владельце по номеру автомобиля.

6. Дана информация о 10 автомобилях. Структура имеет следующие поля: марка, тип кузова, модель, номер, цвет, год выпуска, пробег, цена. Вывести информацию о самом дорогом и самом дешевом автомобиле.

7. Дана информация о 10 рабочих цеха. Структура имеет следующие поля: фамилия, имя, отчество, дата рождения (год, месяц, число), домашний адрес (улица, дом, квартира), оклад, стаж работы. Вывести данные о рабочем с наибольшей зарплатой и наименьшим стажем.

8. Дана информация о 10 абитуриентах. Структура имеет следующие поля: фамилия, имя, отчество, дата рождения (год, месяц, число), номер телефона, домашний адрес (почтовый индекс, область, город, улица, дом, квартира), средний балл аттестата, данные о ЕГЭ (предмет и балл). Вывести информацию о студентах с наибольшим суммарным баллом по ЕГЭ.

9. Дана информация о 10 странах. Структура имеет следующие поля: название страны, столица, государственный язык, площадь территории, денежная единица. Вывести информацию о странах, площадь которых меньше средней площади рассматриваемых стран.

10. Дана информация о 10 фильмах. Структура имеет следующие поля: название, режиссер (имя, фамилия), год выхода, страна, бюд-

жет, кассовые сборы. Вывести информацию о трёх фильмах с наибольшей прибылью.

11. Дана информация о 10 книгах. Структура имеет следующие поля: название, автор (фамилия; имя), год выхода, издательство, количество страниц, себестоимость, цена. Вывести информацию о книге с наибольшей прибылью и наименьшим количеством страниц.

12. Дана информация о 10 рейсах транспортной компании. Структура имеет следующие поля: марка автомобиля, номер автомобиля, пункт назначения, грузоподъемность (в тоннах), стоимость единицы груза; общая стоимость груза. Вывести информацию о рейсах, в которых стоимость груза выше среднего.

13. Дана информация о 10 товарах. Структура имеет следующие поля: наименование, стоимость, срок хранения, сорт, дата выпуска, срок годности. Вывести информацию о товарах с наименьшей стоимость и наибольшим сроком годности.

14. Дана информация о 10 зданиях. Структура имеет следующие поля: адрес (улица, номер дома), количество этажей, количество квартир, срок эксплуатации, срок до капитального ремонта. Вывести информацию о зданиях в порядке возрастания срока до капитального ремонта.

15. Дана информация о 10 квартирах. Структура имеет следующие поля: владелец (фамилия, имя, отчество), телефон владельца, *e-mail* владельца, площадь, число комнат, этаж. Вывести данные о квартирах с площадью меньше 50 кв. м. и подсчитать их количество.

16. Дана информация о 10 фирмах. Структура имеет следующие поля: название фирмы, директор (фамилия, имя, отчество), количество сотрудников, адрес, уставной капитал. Вывести информацию о фирмах с уставным капиталом ниже среднего.

17. Дана информация о 10 читателях библиотеки. Структура имеет следующие поля: читатель (фамилия, имя, отчество), год рождения, адрес (улица, номер дома, номер квартиры), телефон, номер читательского билета, количество книг на руках. Вывести фамилию и телефон читателя с наименьшим количеством книг на руках.

18. Дана информация о 10 кораблях. Структура имеет следующие поля: тип корабля, длина корпуса, ширина корпуса, водоизмещение, скорость, численность экипажа. Вывести список кораблей с численностью экипажа больше заданной.

19. Дана информация о 10 кафедрах вуза. Структура имеет следующие поля: название кафедры, название факультета, список направлений подготовки, численность студентов. Вывести кафедру с наибольшей численностью студентов.

20. Дана информация о 10 экзаменах. Структура имеет следующие поля: название дисциплины, группа, дата, время, аудитория, экзаменатор. Вывести список экзаменов по заданной дате.

Лабораторная работа 5

ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СПЕЦИАЛЬНЫХ СТРУКТУР ДАННЫХ

Цель: изучить алгоритмы работы с динамическими структурами данных в виде стека.

Краткая теория

Стек (*Stack*) – структура типа ***LIFO*** (*Last In, First Out*) – последним вошел, первым выйдет. Элементы в стек можно добавлять или извлекать только через его вершину. Программно стек реализуется в виде однонаправленного списка с одной точкой входа – вершиной стека.

Максимальное число элементов стека не ограничивается, т. е. по мере добавления в стек нового элемента память под него должна запрашиваться, а при удалении – освобождаться. Таким образом, стек – динамическая структура данных, состоящая из ***переменного*** числа элементов.

При работе со стеком обычно выполняются следующие операции:

- формирование стека (добавление элемента в стек);
- обработка элементов стека (просмотр, поиск, удаление);
- освобождение памяти, занятой стеком.

Пример. Реализации стека с помощью одномерного массива. Стек можно реализовать в виде следующей структуры:

```
#define SIZE 100
struct Stack {
    float elem[SIZE];
    int top;
};
```

где *SIZE* – максимальное количество элементов в стеке;

elem – массив из *SIZE* чисел типа *float*, предназначенный для хранения элементов стека;
top – индекс элемента, находящегося в вершине стека.

Для инициализации стека может быть использована следующая функция (индекс элемента, находящегося в вершине стека, равен 0):

```
void initStack(struct Stack *stack) {
    stack->top = 0;
}
```

Получение верхнего элемента стека без его удаления:

```
float stackTop(struct Stack *stack) {
    if((stack->top) > 0) {
        return stack->elem[stack->top-1];
    } else {
        printf("Стек пуст!\n");
        return 0;
    }
}
```

Получение общего количества элементов стека:

```
int getCount(struct Stack *stack) {
    return stack->top;
}
```

Проверка стека на пустоту**:** если количество элементов в стеке равно 0, то стек пуст (возвращается 1).

```
int isEmpty(struct Stack *stack) {
    if(stack->top == 0)
        return 1;
    else return 0;
}
```

Варианты заданий

Написать программу по созданию, добавлению, просмотру и решению приведенных далее задач для однонаправленного линейного списка типа *Stack*. Реализовать сортировку стека.

Во всех заданиях создать список из положительных и отрицательных случайных целых чисел. Решение поставленной задачи описать в виде блок-схемы.

1. Разделить созданный список на два: в первом – положительные числа, во втором – отрицательные.

2. Удалить из созданного списка элементы с четными числами.

3. Удалить из созданного списка отрицательные элементы.

4. В созданном списке поменять местами крайние элементы.

5. Перенести из созданного списка в новый список отрицательные элементы, кратные 3.

6. Из созданного списка удалить элементы, заканчивающиеся на цифру 5.

7. В созданном списке поменять местами элементы, содержащие максимальное и минимальное значения.

8. Перенести из созданного списка в новый список все элементы, находящиеся между вершиной и максимальным элементом.

9. Перенести из созданного списка в новый список все элементы, находящиеся между вершиной и элементом с минимальным значением.

10. В созданном списке удалить все элементы, находящиеся между минимальным и максимальным элементами.

11. В созданном списке определить количество элементов, имеющих значения, меньше среднего значения от всех элементов, и удалить эти элементы.

12. В созданном списке вычислить среднее арифметическое и заменить им первый элемент.

13. В созданном списке вычислить среднее арифметическое отрицательных элементов и заменить им первый элемент.

14. Созданный список разделить на два: в первый поместить четные, а во второй – нечетные числа.

15. В созданном списке определить максимальное значение и удалить его.

16. Созданный список разделить на три: в первый поместить кратные 3, а во второй – кратные 5, в третий – все остальные.

17. Из созданного списка удалить каждый второй элемент.

18. Из созданного списка удалить каждый нечетный элемент.

19. В созданном списке вычислить среднее арифметическое и заменить им все четные значения элементов.

20. В созданном списке вычислить среднее арифметическое и заменить им максимальный отрицательный элемент.

ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ ОТЧЕТА

Документ с отчетом по лабораторной работе должен соответствовать следующим требованиям по форматированию: шрифт Times New Roman, размер 14; межстрочный интервал 1,15; красная строка – отступ 1,25; поля: верхнее и нижнее – 2 см, левое – 3 см, правое – 1,5 см. Интервал между абзацами одного стиля – отсутствует.

Текст отчета по лабораторной работе должен содержать:

– название и цель работы;

– индивидуальное задание;

– блок-схему и псевдокод алгоритма программы;

– код программы и результатов ее выполнения;

– пояснительный текст к программе (описание структуры программы, назначения ее основных переменных, способов реализации отдельных функций и т.д.);

– выводы, которые должны доказывать или оценивать правильность составленной программы или объяснять допущенные ошибки.

2. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ (СРС)

В рамках изучения дисциплины информатика ряд тем и соответствующих им вопросов предусмотрены для самостоятельного изучения студентами.

Тема 1 «Классификация дерева компьютерных наук». Рассмотреть следующие вопросы:

– теория информации;

– теория автоматов;

– теория алгоритмов;

– искусственный интеллект;

– компьютерное зрение;

– эволюционные вычисления;

– генетические алгоритмы.

Тема 2 «Формальное определение алгоритма». Рассмотреть следующие вопросы:

– рекурсивные функции;

– машина Тьюринга;

– нормальные алгоритмы А.А. Маркова;
– понятие сложности алгоритма;
– расчет временной сложности;
– асимптотическая сложность;
– классы сложности алгоритмов.

Тема 3 «Обзор современных процедурных языков программирования и областей их применения». Рассмотреть следующие вопросы:
– структура языка программирования Паскаль;
– структура языка программирования *PureBasic*;
– структура языка программирования ПЛ/1;
– структура языка программирования Рапира;
– структура языка программирования *REXX*;
– структура языка программирования *Go*.

Тема 4 «Обзор современных объектно-ориентированных языков». Рассмотреть следующие вопросы:
– структура языка программирования *C#*;
– структура языка программирования *C++*;
– структура языка программирования *ActionScript*.

Тема 5 «Области и примеры применения рекурсии». Рассмотреть следующие вопросы:
– линейная рекурсия;
– повторительная рекурсия;
– каскадная (древовидная) рекурсия;
– косвенная (взаимная) рекурсия;
– удаленная рекурсия;
– фракталы;
– задачи на графах.

Тема 6 «Современные среды разработки для языка *C*». Рассмотреть следующие вопросы:
– среда программирования *Visual Studio Code*;
– среда программирования *Eclipse*;
– среда программирования *XCode*.

Тема 7 «Отличия языков программирования *C* и *C++*». Рассмотреть следующие вопросы:

– задание аргументов по умолчанию;
– способы динамического выделения памяти;
– ссылки;
– отличия структур в *C* от структур в *C++*;
– отличия структур с C от классов в C++.

Тема 8 «Развитие идей объектно-ориентированного программирования. Обзор технологий на платформе *Microsoft.NET* и *JAVA*. Аспектно-ориентированное программирование в современных языках. Современные паттерны проектирования». Рассмотреть следующие вопросы:

– принципы проектирования *SOLID*: принцип единственной обязанности; принцип открыт/закрыт, принцип разделения интерфейсов и принцип инверсии зависимостей;
– порождающие паттерны: строитель, фабрики (фабричный метод, абстрактная фабрика), прототип;
– структурные паттерны: адаптер, мост, компоновщик, декоратор;
– поведенческие паттерны: цепочка обязанностей, команда, интерпретатор, итератор, посредник, хранитель.

3. КОНТРОЛЬ

Изучение дисциплины завершается экзаменом. Список экзаменационных вопросов:

1. Определение понятия «алгоритм». Свойства алгоритмов. Способы записи алгоритмов (на примере алгоритма Евклида).
2. Краткая характеристика языка *C*. Стандартизация и эволюция языка *C*.
3. Определение понятия «переменная». Имена переменных. Символические константы и директива препроцессора *#define*.
4. Понятие «тип данных». Базовые типы данных и модификаторы типов (на примере типа *int*).
5. Определение понятия «константа». Числовые константы. Определение символических числовых констант в файле *limits.h*
6. Символьные константы. Управляющие последовательности.
7. Строковые константы. Длина строковой константы. Строковые константы и массивы символов.

8. Перечисления и константы перечислимого типа. Анонимные перечисления. Переменные перечислимого типа.

9. Объявления переменных. Инициализация. Модификатор *const*.

10. Определение понятий «операция» и «операнд». Классификация операций по числу операндов. Определение понятия «выражение».

11. Арифметические операции. Операции отношения. Логические операции. Особенности вычисления выражений, содержащих логические операции.

12. Операции инкремента и декремента. Поразрядные (битовые) операции.

13. Операция присваивания и ее особенности в языке *C*. Операции, совмещенные с присваиванием.

14. Тернарная условная операция. Приоритет и ассоциирование операций.

15. Понятие термина «приведение типа». Явное и неявное приведения типов. Корректные приведения типов.

16. Простой оператор. Составной оператор (блок). Локальные переменные. Вложенные блоки.

17. Операторы простого выбора *if* и *if ... else*. Вложенные операторы выбора. Реализация множественного выбора с помощью *else if*.

18. Оператор множественного выбора *switch*. Необходимость использования *break*.

19. Оператор цикла с предусловием *while*. Оператор цикла с предусловием *for*. Оператор цикла с постусловием *do ... while*.

20. Функции языка *C* и модульность программы. Интерфейс и реализация функции.

21. Правила (синтаксис) определения функции. «Минимальная» функция. Свойства функций. Оператор *return*.

22. Аргументы функции и результат выполнения функции. Вызов функции. Передача аргументов по значению. Указатели в качестве аргументов функции.

23. Определение понятия «адрес объекта». Операция получения адреса объекта.

24. Размещение в памяти объектов программы. Размеры участков памяти, выделяемые объектам программы.

25. Адреса массивов и функций. Адресная арифметика. Правила адресной арифметики.

26. Указатели. Нетипизированные указатели. Операция доступа по указателю.

27. Указатели и массивы. Определение массива. Способы задания размера массива.

28. Динамические массивы. Освобождение памяти. Функции *calloc*, *malloc* и *free*.

29. Многомерные прямоугольные массивы. Двумерные массивы неопределенного размера.

30. Многомерные массивы и указатели. Многомерные массивы нерегулярной структуры.

31. Определение понятий *rvalue* и *lvalue*. Особенности использования функций, возвращающих *lvalue*.

32. Указатели на функции. Объявление указателей на функции. Объявление указателя на функцию как типа данных.

33. Определение понятия «структура». Поля (члены) структуры. Определение структуры и определение переменных типа «структура». Инициализация структур.

34. Определение понятия «составное имя».

35. Структуры в качестве аргументов функций и возвращаемых значений.

36. Указатели на структуры и массивы структур. Доступ к членам структуры по указателю.

37. Определение новых типов с помощью *typedef*. Битовые поля в структурах.

38. Объединения (*union*). Отличие объединений от структур.

39. Константные указатели. Варианты использования *const* при объявлении указателей. Функция *main* и аргументы командной строки.

40. Общее правило, разбора сложных объявлений (на примере *int* (*(*(**fun*)())[])();)

Список примерных экзаменационных задач:

1. В одномерном массиве *A* из *n* элементов найти сумму всех элементов и заменить положительные элементы на 10.

2. В одномерном массиве *A* из *n* элементов найти произведение всех элементов и заменить отрицательные элементы на 20.

3. В одномерном массиве *A* из *n* элементов найти максимальный среди всех элементов и заменить нулевые элементы на 30.

4. В одномерном массиве *A* из *n* элементов найти минимальный среди всех элементов и заменить ненулевые элементы на 40.

5. В одномерном массиве *A* из *n* элементов найти среднее значение среди всех элементов и вывести индексы нулевых элементов.

6. В одномерном массиве *A* из *n* элементов найти сумму положительных элементов, а также переставить местами третий и предпоследний элементы.

7. В одномерном массиве *A* из *n* элементов найти количество положительных элементов и индекс минимального элемента.

8. В одномерном массиве *A* из *n* элементов найти произведение положительных элементов и индекс максимального элемента.

9. В одномерном массиве *A* из *n* элементов найти максимальный среди положительных элементов и сформировать массив *B* делением элементов массива *A* на 80.

10. В одномерном массиве *A* из *n* элементов найти минимальный среди положительных элементов и сформировать массив *B* умножением элементов массива *A* на 70.

11. В одномерном массиве *A* из *n* элементов найти произведение отрицательных элементов и умножить отрицательные элементы на 20.

12. В одномерном массиве *A* из *n* элементов найти максимальный среди отрицательных элементов и уменьшить отрицательные элементы на 30.

13. В одномерном массиве *A* из *n* элементов найти минимальный среди отрицательных элементов и увеличить отрицательные элементы на 40.

14. В одномерном массиве *A* из *n* элементов найти среднее значение среди отрицательных элементов и разделить положительные элементы на 40.

15. В одномерном массиве *A* из *n* элементов найти количество нулевых элементов и умножить положительные элементы на 30.

16. В одномерном массиве *A* из *n* элементов найти количество ненулевых элементов и уменьшить положительные элементы на 20.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Колокольникова А.И. Спецразделы информатики: основы алгоритмизации и программирования [Электронный ресурс]: практикум. – М.|Берлин: Директ-Медиа, 2019. – 424 с. – Режим доступа: http://biblioclub.ru/index.php? page=book&id=560695
2. Ильиных А.П. Теория алгоритмов: учебное пособие / Урал. гос. пед. ун-т.– Екатеринбург: 2006. – 149 с.
3. Гринченков Д.В., Гринченков В.Д.. Основы теории алгоритмов [Электронный ресурс]: учеб. пособие для студентов, обучающихся по программе среднего профессионального образования, специальность «Программирование в компьютерных системах». - Новочеркасск: ЮРГПУ (НПИ), 2014. - 120 с. – Режим доступа: http://lib.npi-tu.ru/_scripts/show_book2.php?s=12ffb2a497e8453205994c5e1c2d2922bd&i=12&t=pdf&d=1
4. Иванченко А.Н., Масленников А.А., Иванченко П.А. Основы программирования (язык С): учеб. пособие / Юж.-Рос. гос. политехн. ун-т (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2016. – 88 с.
5. Лубашева Т.В., Железко Б.А. Основы алгоритмизации и программирования [Электронный ресурс]: учеб. пособие. - Минск: РИПО, 2016. - 378 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=463632
6. Волкова Т.И. Введение в программирование [Электронный ресурс]: учеб. пособие. – М.|Берлин: Директ-Медиа, 2018. - 139 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=493677
7. Царев Р.Ю., Прокопенко А.В. Алгоритмы и структуры данных (CDIO) [Электронный ресурс]: учебник. - Красноярск: СФУ, 2016. - 204 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=497016

Учебно-методическое издание

Кущий Дарья Николаевна
Масленников Алексей Александрович

ИНФОРМАТИКА

Методические указания
к лабораторным работам и самостоятельной работе
студентов бакалавриата направлений подготовки
«Математическое обеспечение и администрирование
информационных систем»,
«Информатика и вычислительная техника»,
«Программная инженерия» и «Прикладная математика»

Редактор *Н.А. Юшко*

Подписано в печать 15.07.2022
Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл.-печ. л. 1,86. Уч.-изд. 2,0. Тираж 50 экз. Заказ 46-0524.

Южно-Российский государственный политехнический
университет (НПИ) имени М.И. Платова
Редакционно-издательский отдел ЮРГПУ(НПИ)
346428, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Первомайская, 166
idp-npi@mail.ru