Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

А. Б. ГРАДУСОВ Ю. В. ТИХОНОВ

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ

Учебно-практическое пособие

Владимир 2020

УДК 004.421
ББК 32.973
Г75

Рецензенты:

Доктор технических наук, профессор
зав. кафедрой информационных систем и программной инженерии
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
И. Е. Жигалов

Кандидат технических наук
директор научно-производственного предприятия «Энергоприбор»
В. В. Моисеенко

Г75 **Градусов, А. Б.** Программирование и основы алгоритмизации : учеб.-практ. пособие / А. Б. Градусов, Ю. В. Тихонов ; Владим. гос. ун-т. им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2020. – 164 с.
ISBN 978-5-9984-1100-7

Рассмотрены основные сведения об алгоритмах, часто используемых типах данных, а также наиболее распространенных операциях над значениями этих типов.

Предназначено для выполнения практических и лабораторных работ студентами-бакалаврами технических направлений 09.03.03 «Прикладная информатика», 27.03.04 «Управление в технических системах» дневной и заочной форм обучения.

Рекомендовано для формирования общепрофессиональных компетенций в соответствии с ФГОС ВО.

Табл. 132. Ил. 8. Библиогр.: 7 назв.

УДК 004.421
ББК 32.973

ISBN 978-5-9984-1100-7

© Градусов А. Б.,
Тихонов Ю. В., 2020

ПРЕДИСЛОВИЕ

Пособие предназначено для использования в рамках дисциплин «Программирование и основы алгоритмизации» и «Алгоритмизация и программирование».

Основная его цель состоит в изучении базового синтаксиса – одного из наиболее популярных языков программирования общего назначения – языка Си/Си++ и базовых приемов составления алгоритмов.

Изложенный материал разделен на семь глав, посвященных одному из понятий программирования. Каждая глава содержит изложение теоретических положений, набора вопросов для самоконтроля, практических заданий для самостоятельной работы и описание практической работы по индивидуальным заданиям. Последовательность рассмотрения материала позволяет, начиная с первой главы, приступить к практическому программированию.

При выполнении индивидуальных практических заданий непосредственно на персональном компьютере (ПК) необходимо реализовать разработанные алгоритмы в среде Microsoft Visual Studio C++ и продемонстрировать результаты работы преподавателю.

В качестве результата изучения материала предполагается формирование у обучающихся ряда профессиональных компетенций, в том числе способность применять в профессиональной деятельности языки и технологии программирования, разработку алгоритмических и программных решений профессиональных задач.

Описание синтаксиса и семантики языка Си/Си++ приводится в минимальном объеме, более полное описание можно найти в [1 – 7].

Глава 1. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ

1.1. Классификация языков программирования

В настоящее время существует огромное количество самых разных языков программирования. По некоторым подсчетам, всего их более ста. Некоторые из них очень распространены и популярны (Си++, Java, Python), некоторые известны только узкому кругу специалистов и любителей экзотики (Groovy, Clojure, Boo). Есть несколько подходов к классификации языков программирования. Их можно разделять между собой по целому ряду признаков. Рассмотрим самые известные.

По степени абстракции от машины можно выделить две группы языков: языки низкого уровня и языки высокого уровня.

Язык низкого уровня – язык программирования, близкий к программированию непосредственно в машинных кодах используемого процессора. Такие языки имеют простые машиноподобные команды и осуществляют прямой доступ к памяти.

Язык высокого уровня – язык программирования, разработанный для быстроты и удобства использования программистом. Основная черта высокоуровневых языков – это абстракция, то есть введение смысловых конструкций, кратко описывающих такие структуры данных и операции над ними, описания которых на машинном коде очень длинны и сложны для понимания. Языки предоставляют возможность определять сложные структуры данных, доступ к памяти осуществляется через операции.

На рисунке 1.1 представлена диаграмма, на которой обозначена принадлежность популярных языков программирования к высокому и низкому уровням.

Основное достоинство алгоритмических языков высокого уровня – возможность описания программ решения задач в форме, максимально удобной для восприятия человеком. Но так как каждое семейство ЭВМ имеет свой собственный, специфический внутренний (машинный) язык и может выполнять лишь те команды, которые записаны на этом языке, то для перевода исходных программ на машинный язык используются специальные программы-трансляторы. Работа

всех трансляторов строится по одному из двух принципов: интерпретация или компиляция. В зависимости от того, какой принцип трансляции использует язык программирования, они делятся на компилируемые, интерпретируемые языки и байт-код.

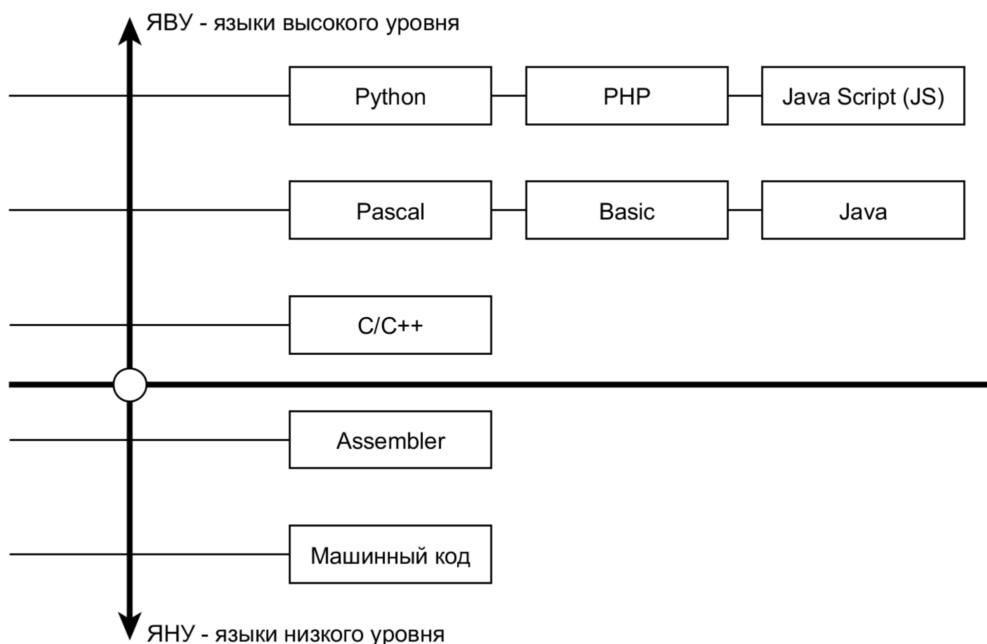

Рисунок 1.1 – Языки программирования низкого и высокого уровней

Компилируемый язык – язык программирования, исходный код которого преобразуется компилятором в машинный код и записывается в файл, с особым заголовком и/или расширением, для последующей идентификации этого файла, как исполняемого.

Интерпретируемый язык – язык программирования, в котором исходный код программы не преобразовывается в машинный код для непосредственного исполнения процессором, а исполняется с помощью специальной программы-интерпретатора.

Байт-код – стандартное промежуточное представление, в которое может быть переведена программа автоматическими средствами. Является промежуточным звеном между исходным кодом и машинным кодом. Примером языка использующего байт-код является Java.

На рисунке 1.2 приведена схема иллюстрирующая принадлежность наиболее популярных языков программирования высокого уровня к интерпретируемым и компилируемым.

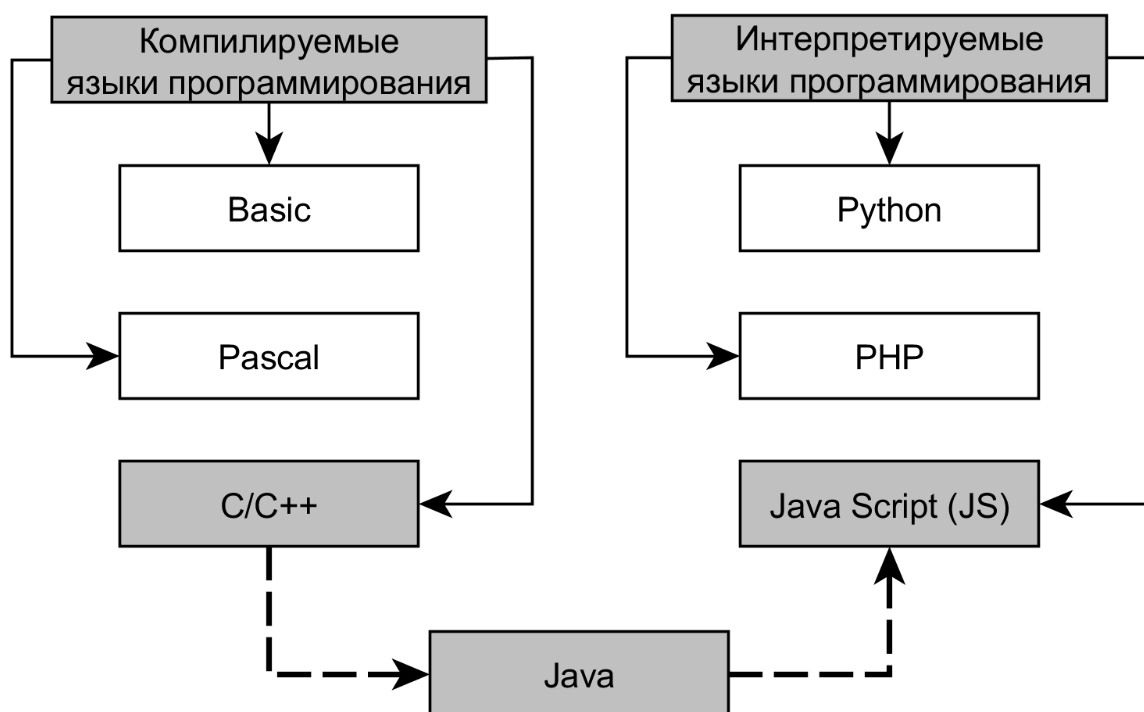

Рисунок 1.2 – Интерпретируемые и компилируемые языки программирования

Одним из важнейших признаков классификации языков программирования является принадлежность их к одной из *парадигм* (концепций): процедурное программирование, объектно-ориентированное программирование (ООП), функциональное программирование, логическое программирование, параллельное программирование.

Классическое процедурное программирование требует от программиста детального описания того, как решать задачу, т. е. формулировки алгоритма и его специальной записи. Основные понятия языков этих групп – оператор и данные. При процедурном подходе операторы объединяются в группы – процедуры.

В рамках этой парадигмы существует структурное программирование.

Структурное программирование – методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. В соответствии с данной методологией любая программа состоит из трёх базовых управляющих структур: последовательность, ветвление и цикл.

Структурное программирование подразумевает точно обозначенные управляющие структуры, программные блоки, отсутствие инструкций GOTO, автономные подпрограммы, в которых поддерживается рекурсия и локальные переменные. Главным в структурном программировании является возможность разбиения программы на составляющие ее элементы. К структурным языкам относятся языки Паскаль, Си.

Принципиально иное направление в программировании связано с методологиями непроцедурного программирования. К ним можно отнести объектно-ориентированное программирование.

Объектно-ориентированное программирование – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Объектно-ориентированное программирование базируется на трех базовых понятиях: инкапсуляция, наследование и полиморфизм. Из языков объектно-ориентированного программирования, популярных среди профессионалов, следует назвать прежде всего Си++, для более широкого круга программистов предпочтительны языки типа Delphi и Visual Basic.

1.2. Понятие алгоритма

Решить задачу – это значит получить результат, отвечающий целям данной задачи. Процесс решения представляет собой совокупность вполне определенных действий над исходными данными. В большинстве случаев эта совокупность действий может быть задана в виде инструкции настолько подробно, что ее исполнение становится механическим процессом. Такая инструкция называется *алгоритмом.*

Понятие алгоритма имеет строгое математическое определение, но мы остановимся на интуитивно-содержательном определении: алгоритмом называется последовательность точных и понятных действий (предписаний, правил, инструкций, команд) над данными, приводящая к получению результата решения любой конкретной задачи из некоторого класса однотипных задач.

Основные свойства любого алгоритма:

детерминированность, означающая, что применение алгоритма к одним и тем же исходным данным должно приводить к одному и тому же результату;

массовость, позволяющая получать результат при различных исходных данных;

результативность, обеспечивающая получение результата через конечное число шагов.

Существует несколько способов записи алгоритмов, отличающихся друг от друга наглядностью, компактностью, степенью формализации и другими показателями. Наибольшее распространение получили способы: словесный, в виде блок-схем, псевдокод, в виде программ для ЭВМ.

Словесное описание алгоритма – текст, записанный на естественном языке (например, на Русском). Пример: если у Миши 10 яблок, то у Коли 5 яблок, иначе у Коли 7 яблок.

Блок-схема – графическое представление алгоритма. Пример блок-схемы приведен на рисунке 1.3.

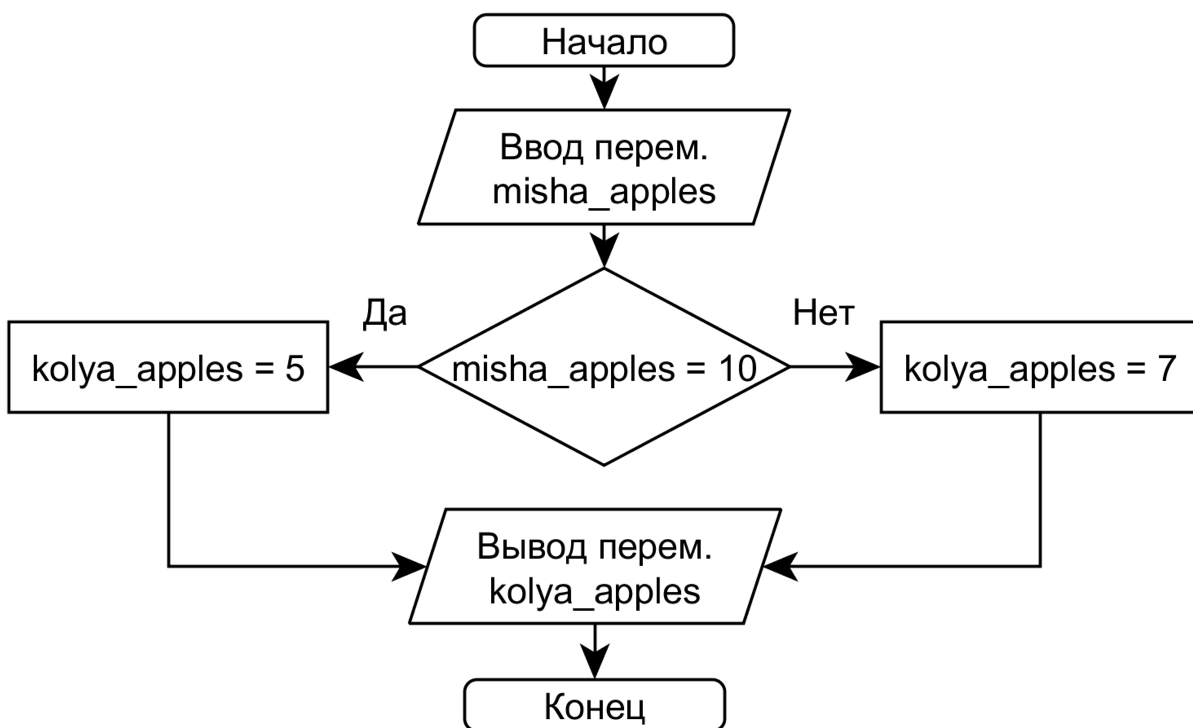

Рисунок 1.3 – Пример блок-схемы

Псевдокод – язык описания алгоритмов, использующий ключевые слова определенного языка программирования (например, Си/Си++) в понятной для человека форме без строгого соблюдения синтаксиса языка.

В таблице 1.1 приведен пример псевдокода.

Таблица 1.1. Пример псевдокода

Код	Комментарии
input(misha_apples)	// ввод данных
if (misha_apples == 10) kolya_apples = 5	// если условие верно...
else kolya_apples = 7	// иначе...
output(kolya_apples)	// вывод данных

Программа для ЭВМ – описание алгоритма, записанное на определенном языке программирования при строгом соблюдении синтаксиса языка.

В таблице 1.2 приведен пример кода на языке Си/Си++.

Таблица 1.2. Пример кода на языке Си/Си++

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение библиотек
int main(void) {	// главная функция
int misha_apples; int kolya_apples;	// объявление переменных
scanf("%d", &misha_apples);	// ввод данных
if (misha_apples == 10) kolya_apples = 5;	// если условие истинно...
else kolya_apples = 7;	// иначе...
printf("%d", kolya_apples);	// вывод данных
fflush(stdin); getchar();	// пауза
return 0; }	// завершение программы

1.3. Базовая структура программы на языке Си/Си++

Рассмотрим пример программы на языке Си/Си++, которая выводит на экран сумму чисел (таблица 1.3).

Таблица 1.3. Программа выводящая сумму чисел

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// библиотеки
#define VALUE 7	// иные директивы
int second_value = 3;	// глобальные // переменные

void print_value(int value)
{
  printf("%d", value);
}
	// функции

int main(void)
{
  print_value(VALUE + second_value);
  fflush(stdin);
  getchar();
  return 0;
}
	// главная функция
	// выводит число «10»

Эта программа состоит их следующих структурных элементов: директив, переменных (констант), функций и операторов (инструкций).

Директива – команда препроцессора, служащего для подготовки программы на языке Си/Си++ к компиляции. Директива начинается с символа «#». К наиболее популярным директивам относятся: директива **#define** – служащая, для макроподстановки (замены) и **#include** – служащая, для подключения библиотек.

Библиотека – коллекция функций и классов (в Си++) написанных на исходном языке. Библиотеки бывают стандартными и пользовательскими.

Переменная – адресуемая область памяти, адрес которой можно использовать для осуществления доступа к данным. Данные, находящиеся в переменной (то есть по данному адресу памяти), называются значением этой переменной.

Константа – способ адресации данных, изменение которых в рассматриваемой программе не предполагается или запрещается.

Функция – это фрагмент программного кода (подпрограмма), к которому можно обратиться из другого места программы.

Оператор (инструкция) – это наименьшая автономная часть языка программирования, команда или набор команд.

*Особенности функции **main***

Отдельно следует отметить функцию **main**, т.к. именно с нее начинается выполнение программы. Во время запуска программы, после работы препроцессора и объявления глобальных переменных управление передается функции **main**. После завершения работы функции **main**, завершается и работа программы. Данное правило справедливо только для консольных однопоточных программ, рассматриваемых в настоящем пособии.

Последовательность выполнения операций

На примере функции **main** можно рассмотреть, в каком порядке выполняются операции. После передачи управления функции **main**, все операции заключенные в этой функции выполняются последовательно, шаг за шагом, строка за строкой.

Комментарии, как неотъемлемая часть программы

Комментарии бывают однострочными и многострочными. Однострочный комментарий следует от символов // и до конца строки. Многострочные комментарии следуют от символов /* до символов */ вне зависимости от того, сколько строк находится между ними.

В таблице 1.4 приведен пример программы с комментариями.

Таблица 1.4. Пример программы с комментариями

Код	Комментарии
#include "stdafx.h"// однострочный комментарий #include <stdio.h> // еще один однострочный ком.	однострочные комментарии
/* -- 	Пример многострочного
комментария | --*/	многострочные комментарии
int main(void) // функция main (комментарий) {	Однострочный комментарий
return 0; }	комментариев нет

Правила хорошего тона в программировании на Си/Си++

1. Хороший программист называет переменные:

- осознанно (value, а не abc123);
- одним словом (value, а не value_of_something_variable);
- строчными буквами (value, а не VALUE или Value или VaLuE);
- по-английски (value, а не znachenie);
- без цифр (однако, если они есть – то должны стоять после букв);
- нарушает правила, только если может убедительно доказать целесообразность нарушения.

2. Хороший программист объявляет директивы define:

- осознанно (LENGTH, а не ABC123);
- одним или несколькими словами через символ подчеркивания (LENGTH или LENGTH_OF_HOUSE);
- заглавными буквами (LENGTH а не Length или length);
- по-английски (LENGTH, а не DLINA);
- без использования цифр (однако, если они есть – то должны стоять после букв, например KEY_1);
- не нарушает указанных правил.

3. Хороший программист называет функции:

- осознанно;
- одним или чаще 2 – 3 словами через символ подчеркивания (under_score); или по принципу camelCase (верблюжий стиль);
- при использовании стиля **under_score** – хороший программист использует только строчные буквы;
- при использовании стиля **camelCase** – хороший программист использует заглавные буквы во втором и последних словах (как в примере);
- по-английски;
- без использования цифр (однако, если они есть – то должны стоять после букв, например divide_by_2);
- не нарушает указанных правил.

4. Хороший программист старается писать комментарии:

- в самом начале файла программы (чтобы указать, как называется программа, какая версия и кто ее автор);
- у каждой функции и переменной (чтобы точно дать понять, для чего они используются);
- в тех местах программы, где у ее автора возникли сомнения (рекомендуется отмечать такие места специальным обозначением **// !!!**);
- перед начало какого-либо крупного блока внутри функции;
- хороший программист использует линию **// -----** для визуального разделения программы на отдельные части;
- хороший программист оставляет комментарии, только если в них есть необходимость, т.к. нужны они **не для красоты, а для дела**.

5. Хороший программист никогда (за исключением структуры switch/case, которая будет рассмотрена позднее) **не будет писать программу в одну строку** (таблица 1.5).

Таблица 1.5. Пример записи программы «в одну строку»

Код	Комментарии
int main(void) { printf("hello"); return 0; }	// плохой // пример

Вместо этого он запишет ее так, как показано в таблице 1.6.

Таблица 1.6. Пример корректной записи программного кода

Код	Комментарии

int main(void)
{
  printf("hello");
  return 0;
}
	// хороший
	// пример

6. Хороший программист относится к фигурным скобкам с особым вниманием: он всегда ставит открывающиеся и закрывающиеся скобки на отдельных строках, как показано в таблице 1.7.

Таблица 1.7. Пример корректной расстановки фигурных скобок и пробелов

Код	Комментарии
while (true) {	// <- нет пробелов
i = i + 1; j = j - 1;	// <- 2 пробела
}	// <- снова нет пробелов

Он всегда ставит 2 (или 4) дополнительных пробела во всех строках между открывающейся и закрывающейся скобкой, это относится и к вложенным скобкам (таблица 1.8).

Таблица 1.8. Пример записи программы содержащей вложенные блоки

Код	Комментарии
while(true) {	// нет пробелов
if (true) {	// 2 пробела
for (;;) {	// 4 пробела
i = i + 1;	// 6 пробелов
}	// снова 4 пробела
}	// снова 2 пробела
}	// снова нет пробелов

7. Хороший программист ставит 2 (или 4) пробела и в других случаях: после условий **for, while, if** и **else**. Однако делает он это только в случае, если выражение не помещается в одну строку (таблица 1.9).

Таблица 1.9. Примеры записи программы содержащей операторы if и else

Код	Комментарии

if (mish_apples == 7)
  kolya_apples = 5;
else
  misha_apples = 3;
	// в данном примере
	// перенос строк необходим

if (value == 7) answer = 5; else answer = 3;	// в данном примере удалось // обойтись без переносов

8. Хороший программист, а особенно программист на Си/Си++ **использует строчные и заглавные буквы с умом** ведь Си/Си++ – не прощает таких ошибок, для него имена Value, VALUE и VaLuE – это 3 абсолютно разных имени.

Вопросы для самопроверки

1. Что такое алгоритм?
2. Какие способы записи алгоритмов вы знаете?
3. Си – это компилируемый или интерпретируемый язык?
4. Си – это язык высокого или низкого уровня?
5. Перечислите три базовых управляющих структуры применяемых в структурном программировании.
6. Чем директива отличается от константы?
7. Чем директива отличается от переменной?
8. В чем состоит главная особенность функции main?
9. Как в программе на языке Си/Си++ написать комментарий?
10. Перечислите основные «правила хорошего тона» в программировании.

Практические задания

1. Необходимо написать программу на языке Си выводящую на экран фамилию и инициалы студента.
2. Необходимо написать программу на языке Си выводящую на экран число «5» в виде ASCII-графики (следует использовать символы «*» и «пробел»). Подсказка: перенос строки – это символ «\n».
3. Необходимо написать программу на языке Си, которая выводит на экран число, заданное в виде директивы.
4. Необходимо написать программу на языке Си, которая выводит на экран произведение трех чисел, заданных в виде директив.
5. Необходимо написать программу на языке Си, которая выводит на экран фразу «В моей группе ***N*** студентов». ***N*** – реальное число студентов в группе. Число ***N*** должно быть задано в виде директивы.

Глава 2. ОБРАБОТКА ДАННЫХ. МАТЕМАТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОПЕРАЦИИ

2.1. Типы данных

Любые данные, которые используются для программной обработки в ЭВМ, являются значениями программных объектов (констант, переменных, функций) и характеризуются своими типами.

Тип данных определяет множество допустимых значений, которые может иметь тот или иной программный объект, и множество применимых операций к нему.

Кроме того, тип данных определяет также и формат внутреннего представления данных в ЭВМ.

В одних языках программирования, например, таких как Паскаль, Си/Си++, программный объект связывается с типом в момент объявления и тип не может быть изменен позже.

В других языках программирования, например, Python, переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов.

Язык Си/Си++ характеризуется разветвленной структурой типов. Диаграмма с указанием основных типов данных в Си/Си++ приведена на рисунке 2.1.

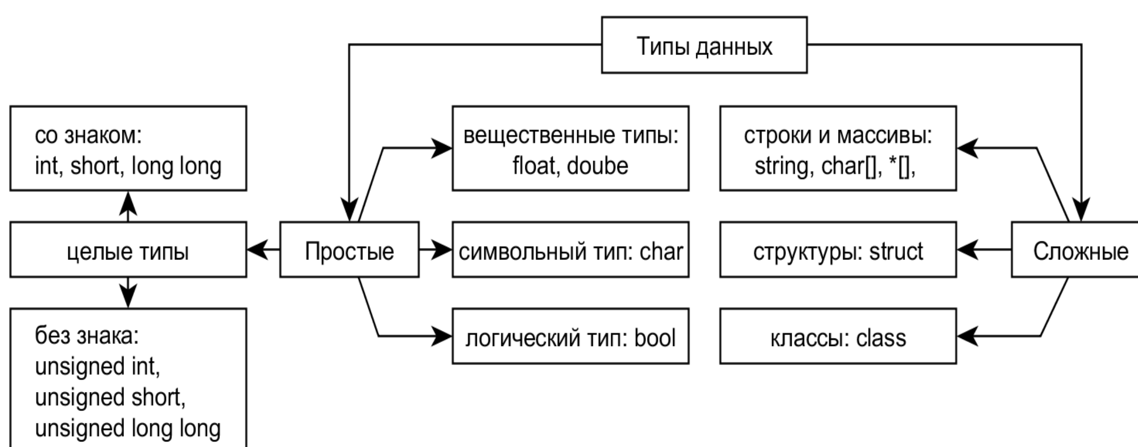

Рисунок 2.1 – Основные типы данных Си/Си++

Типы данных делятся на простые и сложные.

К простым типам данных относятся следующие типы: целочисленный (со знаком и без знака), вещественный (с плавающей запятой), символьный, логический и др. типы.

Сложный тип данных – это тип данных, элементы которого имеют внутреннюю структуру, доступную программисту. Иными словами сложный тип данных включает в себя одновременно несколько элементов простого типа (типов).

Сложные типы данных могут состоять как из элементов одного типа, так и из элементов разных типов. К сложным типам данных относятся: строки, массивы, структуры, классы и др. типы.

2.2. Простые типы данных

Целочисленный тип данных

Диапазон возможных значений целочисленных типов зависит от их внутреннего представления, которое может занимать два, четыре или восемь байт.

Основной тип: **int** (сокращение от integer – целое число). Основные особенности: имеет знак; занимает в памяти 4 байта; может принимать значения в диапазоне от -2147483648 до +2147483647.

«Укороченный тип» **short int** (или просто **short**). Основные особенности: имеет знак, занимает в памяти 2 байта, может принимать значения в диапазоне от -32768 до +32767.

«Удлиненный» тип – **long long** (поддерживается языком Си, начиная со стандарта С99). Основные особенности: имеет знак, занимает в памяти 8 байт, может принимать значения в диапазоне от -9 223 372 036 854 775 807 до +9 223 372 036 854 775 807 (или ±9 квинтильонов).

В качестве примера рассмотрим, как в памяти хранится значение 7 типа **short** (таблица 2.1).

Таблица 2.1. Пример хранения данных в переменной типа **short**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Позиция 0 – это знак (0 – положительный, 1 – отрицательный).

Позиции 1..15 – само число в дополнительном коде (в данном примере $111_2 = 7_{10}$). Если бы знак был единичным, то код соответствовал бы числу $–32761_{10}$.

Особенность: число «ноль» записывается, как 0000000000000000_2, при этом «отрицательного нуля» не существует, а запись 1000000000000000_2 будет означать -32768_{10}. Именно за счет этой особенности диапазон отрицательных чисел (для **short** – это [-32768, -1]), для целочисленных типов со знаком на единицу больше диапазона положительных чисел (для **short** – это: [1, 32767]).

Помимо целочисленных типов данных со знаком, существуют точно такие же типы, но без знака:

unsigned int (диапазон от 0 до 4294967295);

unsigned short (диапазон от 0 до 65535);

unsigned long long (диапазон от 0 до ≈18 квинтильонов).

В качестве примера рассмотрим, как в памяти ЭВМ хранится значение переменной типа unsigned short (таблица 2.2).

Таблица 2.2. Пример хранения данных в переменной типа unsigned short

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Позиции 0..15 – само число в двоичном коде (в данном примере $111_2 = 7_{10}$).

Особенность: основным преимуществом без знакового типа является то, что в переменных такого типа можно хранить вдвое большие положительные числа, например, в переменных типа **short** можно хранить числа до +32767, а в переменных типа **unsigned short**, до +65535.

Пример программы с целочисленными переменными приведен в таблице 2.3.

Таблица 2.3. Программа, работающая с целочисленными переменными

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение // библиотек
int main(void) {	// основная функция

short small_value = 1000;
int medium_value = 100000;
long long big_value = 1000000000000;
	// простые примеры

short small_value_two;	// в «пустой» ячейке
	// может храниться либо 0
	// либо случайное число

int first_question = 75.9; int second_question = 6.1;	// запишется 75 // запишется 6
return 0; }	// завершение // программы

Вещественный тип данных

Значениями вещественного типа являются вещественные числа.

Если в математике вещественные числа – это бесконечное непрерывное множество, то в программировании ему соответствует конечное множество вещественных чисел, т.к. в памяти ЭВМ могут быть представлены числа только из конечного подмножества вещественных чисел.

В памяти ЭВМ вещественные числа хранятся в форме с плавающей запятой.

Число с плавающей запятой (или число с плавающей точкой) – форма представления вещественных (действительных) чисел, в которой число хранится в форме мантиссы и показателя степени.

В общем случае, значение переменной с плавающей запятой определяется по формуле:

$$A = Ms \cdot M \cdot 10^{Ns \cdot N} = \pm M \cdot 10^{\pm N},$$

где Ms – знак мантиссы, M – мантисса числа, Ns – знак порядка, N – порядок числа.

Абстрактный пример хранения числа с плавающей запятой в памяти компьютера приведен в таблице 2.4.

Таблица 2.4. Абстрактный пример хранения числа с плавающей запятой

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	1	0	1	0	0	0	0	0	1	1
	M – мантисса								N – степень						
Ms – знак								Ns – знак степени							

Позиция 0 – знак (0 – положительный, 1 – отрицательный), в пример: «+».

Позиции 1..7 – мантисса (в примере $110010_2 = 50_{10}$).

Позиция 8 – знак степени (аналогично с позицией 0), в примере: «-».

Позиции 9..15 – степень (в примере $11_2 = 3_{10}$).

В нашем примере (таблица 2.4) получаем:

$$A = +50 \cdot 10^{-3} = +0.05.$$

Запись чисел с плавающей запятой возможна в классической и экспоненциальной форме.

Экспоненциальная запись – представление действительных чисел в виде мантиссы и порядка. Такая запись удобна при представлении очень больших и очень малых чисел, а также для унификации их написания.

В Си/Си++ существует два основных типа переменные с плавающей запятой:

- **float** – имеет знак, занимает в памяти 4 байта, имеет диапазон положительных значений: от $1.1755 \cdot 10^{-38}$ до $3.4028 \cdot 10^{38}$.
- **double** – имеет знак, занимает в памяти 8 байт, имеет диапазон положительных значений: от $2.2250 \cdot 10^{-308}$ - $1.7977 \cdot 10^{308}$.

Главным недостатком типа **float** является малая длина мантиссы – всего 23 бита. При этом «мантисса» целочисленной переменной, например типа **int** составляет 31 байт. Нетрудно подсчитать, что такая разница (в 8 бит) приводит к «падению точности» числа типа **float** по сравнению с типом **int** в 256 раз.

Пример программы с вещественными переменными приведен в таблице 2.5.

Таблица 2.5. Программа, работающая с вещественными переменными

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение // библиотек
int main(void) {	// основная функция

float first_value = 1.9;
float second_value = 5.1;
float third_value = 10;
	// классическая
	// запись
	// запишется 10.0

float fourth_value = 2e5;
float fifth_value = 2.2e-3;
double big_value = 1e100;
	// экспоненциальная
	// запись
	// запишется 10^{100}

return 0; }	// завершение // программы

Символьный тип данных

Существует особый целочисленный тип **char** (сокращение от character – символ) занимающий в памяти 1 байт и способный принимать одно из 256 возможных значений. В переменных такого типа можно хранить символы из таблицы ASCII.

ASCII (American standard code for information interchange) – название таблицы, в которой некоторым печатным и непечатным символам сопоставлены числовые коды. Таблица была разработана и стандартизована в США, в 1963 году.

Переменной типа **char** можно присвоить значение двумя способами: числовым и непосредственно символьным. Для лучшего понимания механизма, рассмотрим пример (таблица 2.6).

Таблица 2.6. Программа, работающая с символьными переменными

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение // библиотек
int main(void) {	// основная функция
char character = 'f'; char second_character = '!';	// присвоение символа
char numeral = 102; char second_numeral = 33;	// присвоение числа
return 0; }	// завершение // программы

Логический тип данных

Существует специальный тип **bool** (сокращение от слова boolean). Переменные этого типа занимают в памяти 1 байт и принимают одно из двух возможных значений **истина** или **ложь**.

Существует два способа присвоить значение переменной типа bool. Первый способ: цифрой – 0 означает ложь, а 1 или любое другое число – истину. Второй способ: специальными ключевыми словами **true** или **false**.

Пример программы с логическими переменными приведен в таблице 2.7.

Таблица 2.7. Программа, работающая с символьными переменными

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение // библиотек
int main(void) {	// основная функция

bool first_num = 0;
bool second_num = 1;
bool third_num = 2;
	// имеет значение false
	// имеет значение true
	// имеет значение true

bool value_yes = true; bool value_no = false;	// присвоение истинного… // … и сложного значений
return 0; }	// завершение // программы

2.3. Ввод и вывод данных в консоль

В общем виде, основные функции для работы с консолью представлены на рисунке 2.2.

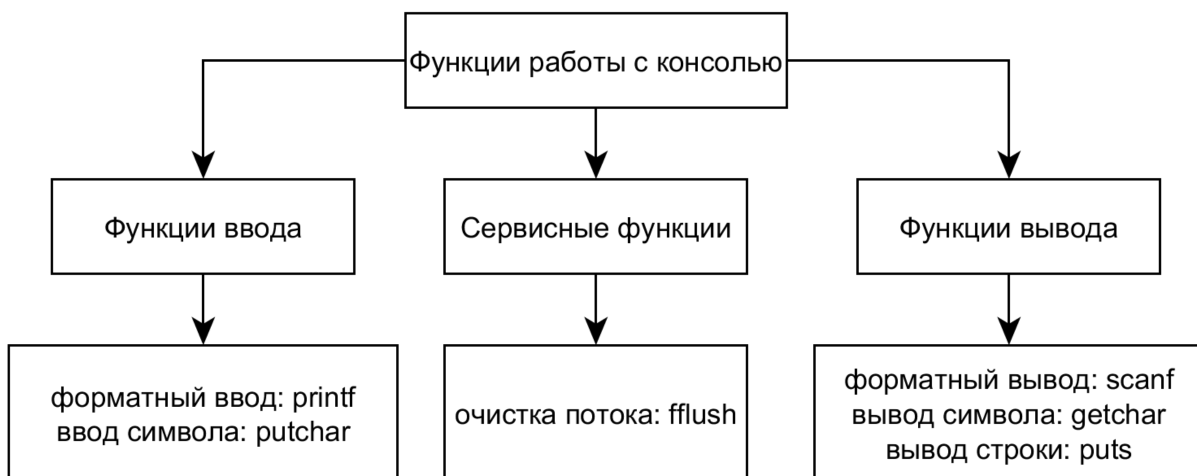

Рисунок 2.2 – Функции работы с консолью

Функция ***printf()*** *и ее применение для вывода текста*

Printf() (print formatted) вывод по формату, иначе говоря – вывод информации в соответствии с шаблоном. Наиболее простым применением этой функции является вывод простого текста на экран. Программа для вывода текста в простейшем случае выглядит следующим образом (таблица 2.8).

Таблица 2.8. Пример использования функции printf()

<table>
<tr><th>Код</th><th>Комментарии</th></tr>
<tr><td>#include "stdafx.h"
#include <stdio.h></td><td>// подключение
// библиотек</td></tr>
<tr><td>int main(void)
{</td><td>// основная функция</td></tr>
<tr><td>printf("-----------------\n");
printf("| Hello World |\n");
printf("-----------------\n");</td><td>// Вывод текста в
// три строки</td></tr>
<tr><td>fflush(stdin);
getchar();</td><td>// пауза</td></tr>
<tr><td>return 0;
}</td><td>// завершение
// программы</td></tr>
</table>

*Спецсимвол **\n** и примеры его использования*

Особый интерес представляет специальный символ **\n**. Этот символ используется для переноса строки. В одной строке программы можно поставить сразу несколько символов переноса.

В таблице 2.9 приводятся примеры использования спецсимвола переноса **\n**.

Таблица 2.9. Пример использования спецсимвола переноса строки

Код	Комментарии
printf("---\nhello world\n---\n");	// --- // hello world // --- // <пустая строка>

printf("---\nhello\nworld\n---\n");	// ---
	// hello
	// world
	// ---
	// <пустая строка>

Вывод данных простых типов

Для вывода данных, функция **printf()** имеет несколько специальных конструкций:

- **%d** (decimal) – вывод переменных типов **int, short**;
- **%u** (unsigned decimal) – вывод переменных типов **unsigned int, unsigned short**;

- **%x** (hex – 16-и ричное) – вывод hex-кода любой целочисленной переменной;
- **%X** (HEX) – аналог предыдущей конструкции, но с заглавными буквами (например, «DA» вместо «da»);
- **%f** (floating point) – вывод переменных типов **float, double**;
- **%c** (character) – вывод символов типа **char**.

В таблице 2.10 представлен пример вывода переменной типа **float**, с указанием и без указания количества знаков после запятой. В этом примере, вместо числа 2 можно указать любое другое, от нуля включительно.

Таблица 2.10. Пример форматированного вывода переменной типа **float**

Код	Комментарии
float value = 5.5;	// объявление переменой и // присвоение ей значения
printf("%f", value);	// вывод переменной // «как есть»
printf("%.2f", value);	// вывод переменной с // двумя знаками после запятой

При выполнении функции **printf()** иногда встает задача, вывести на экран целую часть от переменной с плавающей запятой. Для этого можно воспользоваться преобразованием типов (таблица 2.11).

Таблица 2.11. Пример преобразования типов при работе с функцией printf()

Код	Комментарии
float float_value = 5.5; int int_value = float_value;	// объявление переменных и // присвоение ем значений

printf("%d\n", int_value);	// простой вывод с
	// предварительным
	// преобразованием типов

printf("%d\n", (int)float_value); float round_value = (int)(5.5);	// преобразование типов // «на лету»

Одновременный вывод нескольких переменных

Если требуется вывести одновременно несколько переменных, необходимо: во-первых указать их в форматной строке (с помощью

знака %) и во-вторых перечислить необходимые переменные через запятую после форматной строки (таблица 2.12).

Таблица 2.12. Одновременный вывод нескольких переменных

Код	Комментарии
int a = 7; int b = 25;	// объявление переменных и // присвоение им значений
printf("Values: %d, %d", a, b);	// результат: «Value: 7, 25»
printf("Values: %x, %x", a, b);	// результат: «Value: 7, 19»

Использование русского алфавита

По умолчанию Visual Studio C++ не поддерживает русские символы, для того, чтобы избавиться от этого недостатка, необходимо подключить библиотеку **locale.h** и в начале программы вызывать функцию **setlocale** с параметрами **0, "Russian"**(таблица 2.13).

Таблица 2.13. Пример использования русского алфавита

Код	Комментарии
#include "stdafx.h" #include <stdio.h> #include <locale.h>	// подключение библиотек
int main(void) {	// основная функция
setlocale(0, "Russian");	// включение русской таблицы
printf("Здравствуй мир!");	// вывод русского текста
fflush(stdin); getchar();	// пауза
return 0; }	// завершение программы

Ввод данных с помощью функции scanf( )

Функция ввода данных **scanf()** в целом аналогична функции **printf()**, но используется для ввода данных, при этом требует указания не самих переменных, а их адресов. На практике это выглядит следующим образом (таблица 2.14).

Таблица 2.14. Пример использования функции scanf( )

Код	Комментарии
int value;	// объявление переменной
scanf("%d", &value);	// следует обратить внимание // на знак амперсанта

Важная особенность: при работе в современной среде разработки, например в Visual Studio 2015, для нормальной работы функции **scanf()** может потребоваться произвести дополнительные действия: отредактировать файл **stdafx.c** согласно с образцом (таблица 2.15).

Если файл **stdafx.c** отсутствует, см. Приложение.

Таблица 2.15. Директива, отключающая предупреждения CRT

Код	Комментарии
#define _CRT_SECURE_NO_WARNINGS	// отключение системы // предупреждений CRT
#include "stdafx.h"	// данная строка имеется // по умолчанию

Одновременный ввод нескольких переменных

При одновременном вводе нескольких переменных можно использовать любые разделяющие символы, например пробел или запятую (таблица 2.16).

Таблица 2.16. Пример одновременного ввода нескольких переменных

Код	Комментарии
int value_1; int value_2;	// объявление // переменных
scanf("%d,%d", &value_1, &value_2);	// ввод через запятую
scanf("%d %d", &value_1, &value_2);	// ввод через пробел

Дополнительные функции ввода и вывода

Помимо функций **printf()** и **scanf()** в языке Си/Си++ имеется ряд других функций ввода/вывода. Рассмотрим три наиболее популярные функции:

Функция **puts()** используется, для вывода простой строки на экран, без возможности использования форматирования. Преимуще-

ством этой функции над **printf()** является большая скорость ее выполнения.

Функция **putchar()** используется для вывода одного символа типа char на экран. Эта функция отличается от **printf()** простотой применения.

Функция **getchar()** используется для ввода одного символа типа char. Данная функция может быть полезной при создании меню. Примеры использования дополнительных функций ввода/вывода приведены в таблице 2.17.

Таблица 2.17. Дополнительные функции ввода/вывода

Код	Комментарии
char value = 'X';	// объявление // переменной
printf("hello world\n"); puts("hello world\n");	// преимущество puts // в скорости
printf("%c", value); putchar(value);	// преимущество putchar // в простоте
scanf("%c", &value); value = getchar();	// преимущество putchar // в простоте

getchar();	// особый пример, в
	// котором getchar
	// используется для
	// остановки программы

*Функция очистки буфера ввода **fflush( )***

После завершения ввода данных, рекомендуется проводить очистку буфера чтения. Это необходимо для того, чтобы игнорировать случайные данные, которые поступили в программу после ввода полезной информации. Например, вы ввели какое-то число, а затем несколько раз нажали на клавишу **Enter**. Это приводит к поступлению в программу нежелательных данных (в данном примере – символов переноса строки).

Чтобы избавиться от этих ненужных данных существует функция **fflush()**. Данную функцию необходимо запускать с параметром **stdin**. Пример использования функции показан в таблице 2.18.

Таблица 2.18. Пример использована функции fflush()

Код	Комментарии
printf("hello world");	// вывод данных
fflush(stdin);	// очистка буфера чтения
getchar();	// пауза: ждем, пока // пользователь нажмет на кнопку

Данная функция поддерживается не во всех версиях Visual Studio, по этой причине на нее полагаться не следует, однако если эта функция присутствует и работает нормально – она может существенно упростить написание программы.

Если эта функция отсутствует, то рекомендуется в конце программы вставлять строку **getchar();** дважды.

2.4. Простые операции обработки данных, математические и логические операции

Простые операции обработки данных в языке программирования Си/Си++ делятся на 4 основных типа: целочисленные математические операции, вещественные математические операции, логические операции, атомарные операции (рисунок 2.3). В этом разделе речь пойдет о первых трех типах операций.

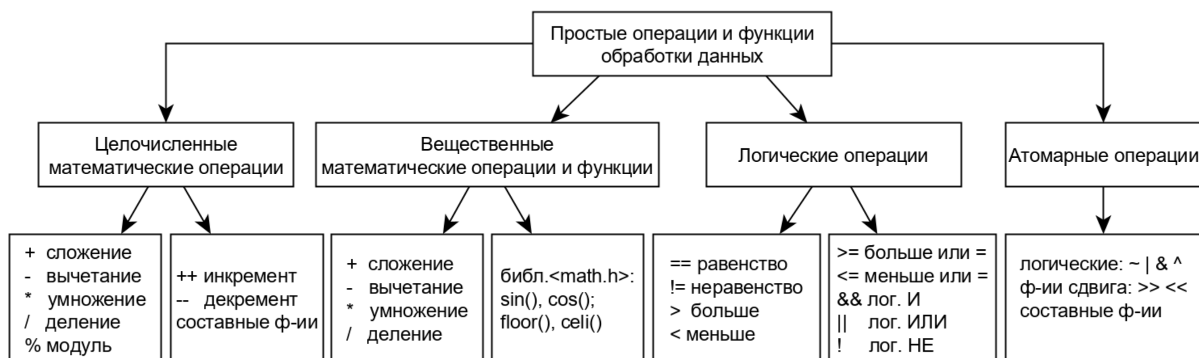

Рисунок 2.3 – Простые операции и функции языка Си/Си++

Операции с целочисленными переменными

При работы с целочисленными переменными (int, short, unsigned int и др.) существует пять базовых и семь дополнительных операций. Базовые операции – это сложение (+), вычитание (-), умножение (*), деление (/) и остаток от деления или модуль (%). Следует отметить, что в Си/Си++ на ноль делить нельзя!

В таблице 2.19 приводится пример программы – калькулятора использующего все приведенные выше функции ввода/вывода.

Таблица 2.19. Программа калькулятор

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение библиотек
int main(void) {	// основная функция
int first, second; int sum, sub, mul, div;	// объявление переменных
scanf("%d %d", &first, &second);	// ввод целого числа

sum = first + second;
sub = first - second;
mul = first * second;
div = first / second;
	// обработка данных

printf("Sum = %d\n", sum);
printf("Sub = %d\n", sub);
printf("Mul = %d\n", mul);
printf("Div = %d\n", div);
	// вывод данных

fflush(stdin); getchar();	// пауза
return 0; }	// завершение программы

Следует отдельно рассмотреть функцию модуля. Дело в том, что при работе с целочисленными переменными в результате выполнения операции деления может остаться остаток. Для того чтобы определить остаток от деления служит операция модуля (%). Данная операция широко используется при разработке различных алгоритмов, о чем будет сказано в последующих главах.

Рассмотрим приведенный выше пример (таблица 2.19). Как мы видим, в программе имеются две переменные, значения которых вводятся с клавиатуры: first и second. После ввода, эти переменные используются только справа от оператора присвоения «=». Как вы думаете, можно ли использовать одну и ту же переменную сразу справа и слева от оператора «=»? И можно ли использовать просто «числа» (константы) вместо переменных? Ответ приводится в таблице 2.20.

Таблица 2.20. Простейший пример работы с переменными

Код	Комментарии
int first, second = 7;	// объявление переменных
first = second + 7;	// пример с константой
second = second + 5;	// пример с одной переменной
first = (first + 1) * first; second = (first + second) * 2;	// пример работы с одной и той же // переменной

Необходимо четко понимать, что сначала выполняются операции по правую сторону знака равенства и лишь затем, происходит запись нового значения в переменную, т.е. сначала происходит сложение переменной second и константы 5. При этом переменная second никак не меняется и только потом, когда дело доходит до операции присвоения (равно), происходит перезапись переменной.

Для упрощения выполнения операций существует семь составных функций, которые приведены в таблице 2.21.

Таблица 2.21. Составные математические функции

Составная функция	Обычная функция	Комментарий
value++;	value = value + 1;	// инкремент
value--;	value = value - 1;	// декремент
value+=7;	value = value + 7;	// сложение с накоплением
value-=5;	value = value - 5;	// вычитание с накоплением
value*=3;	value = value * 3;	// умножение с накоплением
value/=2;	value = value / 2;	// деление с накоплением
value%=4;	value = value % 4;	// модуль с накоплением

Наиболее часто используемыми функциями среди составных, являются функции инкремента и декремента.

Операции с вещественными переменными

При работе с вещественными переменными можно использовать тот же набор базовых операций, что и для целых чисел (сложение, вычитание, умножение и деление). Исключение составляет лишь операция вычисления остатка от деления, т.к. при работе с вещественны-

ми числами такое понятие отсутствует. Ниже приведен пример вычисление частного и остатка от деления (таблица 2.22).

Таблица 2.22. Пример деления целочисленных и вещественных переменных

Код	Комментарии
int first_int, second_int; float float_value;	// объявление // переменных
first_int = 5 / 2; second_int = 5 % 2;	// Результат: 2 // Результат: 1
float_value = 5.0 / 2;	// Результат: 2.5

*Функции библиотеки **math.h***

Подключение библиотеки **math.h** значительно расширяет функциональные возможности языка Си/Си++. В таблице 2.23 приводится лишь несколько наиболее популярных функций библиотеки.

Таблица 2.23. Основные функции библиотеки math.h

Код	Комментарии
value = floor(5.7);	// округление в меньшую сторону
value = ceil(5.7);	// округление в большую сторону
value = fabs(-5.7);	// число без знака (модуль)
value = sqrt(25.0);	// корень квадратный

value = sin(x);
value = cos(x);
value = tan(x);
	// тригонометрические функции

Логические операции

Над операндами логического типа (***bool***) определены следующие логические операции, которые обозначаются следующими символами:

НЕ – !;

И – &&;

ИЛИ – ||.

Результат операции определяется таблицей истинности этой операции и является значением логического типа.

В таблице 2.24 приводится пример использования *логических операций И, ИЛИ и НЕ*.

Таблица 2.24. Логические операции над логическими выражениями

<table>
<tr><th>Код</th><th>Комментарии</th></tr>
<tr><td>bool first = true;
bool second;</td><td>// объявление переменных</td></tr>
<tr><td>second = (first && false);
second = (first || false);
second = !first;</td><td>// логическое И
// логическое ИЛИ
// логическое НЕ</td></tr>
</table>

- Логическая операция «И» иначе называется бинарной конъюнкцией;
- Логическая операция «ИЛИ» иначе называется бинарной дизъюнкцией;
- Логическая операция «НЕ» иначе называется унарным отрицанием (инверсией).

Следует отметить, что для логических операций существует четкий порядок выполнения операций:

- Операции в скобках;
- Инверсия (НЕ);
- Конъюнкция (И);
- Дизъюнкция (ИЛИ).

Логические операции широко применяются при реализации ветвления и циклов, о которых рассказывается в следующей главе.

В языках программирования существуют шесть операций сравнения, результатом которых является значение логического типа.

В таблице 2.25 приведены логические операции над числовыми данными.

Таблица 2.25. Логические операции над числовыми выражениями

Код	Комментарии
int first = 5; bool second;	// объявление // переменных

second = (first > 5);
second = (first >= 5);
second = (first < 5);
second = (first <= 5);
second = (first == 5);
second = (first != 5);
	// больше?
	// больше или равно?
	// меньше?
	// меньше или равно?
	// равно?
	// неравно?

second = (7.25 > 7.20);	// пример с типом float

Следует обратить особое внимание на следующие обстоятельства:

1. Операция сравнения никак не воздействует на свои аргументы (т.е. переменная first никак не изменится).

2. Операция сравнения «равно?» не имеет ничего общего с операцией присвоения. Операция сравнения – это вопрос, а присвоение – это ответ.

3. Операция сравнения имеет тип **bool** (логический тип), а ее аргументы могут быть целочисленными или вещественными.

4. Круглые скобки в примере выполняют ту же роль, что и в математике. Их использование в данных примерах хотя и не является обязательным, однако позволяет лучше понять структуру программы и при этом не допустить ошибок.

Операции сравнения можно комбинировать с логическими операциями, например, когда следует определить принадлежность переменной какому-то интервалу или интервалам (таблица 2.26). При этом будет получаться значение логического типа.

Таблица 2.26. Пример совместного использования логических операций

<table>
<tr><th>Код</th><th>Комментарии</th></tr>
<tr><td>int first = 2;
bool second;</td><td>// объявление
// переменных</td></tr>
<tr><td>second = (first > 5) && (first < 10);
second = (first < 3) || (first > 5);</td><td>// принадлежность
// интервалам</td></tr>
</table>

Вопросы для самопроверки

1. В чем состоит преимущество без знаковых целочисленных типов данных?
2. Какой основной недостаток имеет тип **float**?
3. Можно ли в переменной символьного типа хранить числа?
4. Какие значения могут храниться в переменной логического типа?
5. Перечислите основные функции ввода данных (в консоль).
6. Перечислите основные функции вывода данных (из консоли).
7. В чем состоит преимущество форматного ввода/вывода данных?
8. Что будет, если выводить числа с плавающей запятой без указания количества знаков после запятой?

9. Опишите особенность вывода в консоль переменных вещественного типа.

10. Объясните, для чего используется функция **fflush(stdin)**?

11. Перечислите основные целочисленные математические операции.

12. Перечислите основные вещественные математические операции.

13. Перечислите основные логические операции.

14. Перечислите основные функции библиотеки **math.h.**

15. Приведите пример комплексного использования логических операций.

Практические задания

1. Необходимо написать программу на языке Си в которой объявляется 10 переменных. Объявленные переменные должны быть выведены на экран. В переменные нужно записать:

- числа: 1, 70, 4млн, 8.5, 95.4;
- символы: «A», «1», символ переноса строки;
- переменные принимающие «истинное» и «ложное» значение.

2. Необходимо написать программу на языке Си, в которой имеется достаточно переменных типа char, чтобы можно было сохранить в них слово «hello». Затем необходимо ввести символы из консоли и вывести получившееся слово на экран. Важно: строками (в том числе string и char[]) пользоваться запрещается!

3. Необходимо написать программу на языке Си, которая вводит из консоли длину ребра куба (A), а затем выводит на экран объем куба (V) и площадь поверхности куба (S).

4. Необходимо написать программу на языке Си, которая вводит из консоли радиус окружности (R), а затем выводит длину окружности (L) и площадь круга (S).

5. Необходимо написать программу на языке Си, которая вводит из консоли радиусы окружностей ($R1$ и $R2$) имеющих общий цент, а затем выводит на экран площадь кольца ($S3$), внешний радиус которого равен $R1$, а внутренний радиус равен $R2$.

Практическая работа № 1. **Программирование алгоритмов линейной структуры**

Цель работы – освоение порядка работы в среде Visual Studio, овладение практическими навыками разработки и программирования алгоритмов линейной структуры.

Задание к работе

В соответствии с вариантом задания (таблица 2.27), в среде Visual Studio необходимо разработать программу, выполняющую:

- ввод данных из консоли (см. графу «Входные переменные» в таблице 2.27);

- расчет переменных (формулы расчета см. в графе «Расчетные формулы»);

- вывод в консоль исходных данных и результатов расчета.

Таблица 2.27. Варианты заданий к практической работе № 1

№ вар.	Расчетные формулы	Входные переменные
1	$s = \ln(x + \sqrt{x^2 + 1};$ $z = a \cdot x^{-a \cdot x} \sin(b \cdot x).$	x, a, b
2	$s = \dfrac{2 \cdot \cos(x - \pi / 6)}{0{,}5 + \sin^2(y)};$ $z = 1 + \dfrac{b^2}{3 + b^2 / 5}.$	x, y, b
3	$v = a \cdot x^{3/2};$ $z = \dfrac{6 \cdot a}{\sqrt{x} \cdot (4 + 9 \cdot a^2 \cdot x)}.$	x, a
4	$y = a \cdot t^3;$ $z = \dfrac{1}{27 \cdot a^2}((4 + 9 \cdot a^2 \cdot x)^{3/2} - 8).$	x, a
5	$y = e^{-b \cdot t} \sin(a \cdot t + b) + \sqrt{\lvert b \cdot t + a \rvert};$ $s = b \cdot \sin(a \cdot t^2 \cos(2 \cdot t) - 1.$	t, x, a

№ вар.	Расчетные формулы	Входные переменные
6	$s = x^3 \cdot tg^2(x+b)^2 + a;$ $z = \dfrac{b \cdot x^2 - a}{e^{b \cdot x} - 1}.$	x, a, b
7	$y = \dfrac{a \cdot t^3}{1+t^2};$ $z = \dfrac{a \cdot \sin^2(x)}{\cos(x)}.$	t, a, x
8	$y = a \cdot t(t^2 - 1)^{3/2};$ $z = (e^{t \cdot x} + \pi) \cdot t^2 + 1.$	t, a, x
9	$y = a \cdot t \cdot \dfrac{t^2 - 1}{t^2 + 1};$ $z = 2 \cdot a^2 - 0{,}5 \cdot \pi \cdot a^2.$	t, a
10	$z = (x-a)^2 \cdot (x^2 + y^2) - \ln(x);$ $w = \dfrac{\sqrt{x^2 - a^2}}{2}.$	t, a, y
11	$z = \dfrac{\sin(x)}{\sqrt{1 + t^2 \cdot \sin^2(x)}} - t \cdot \ln(t \cdot x);$ $w = e^{-t \cdot x}\sqrt{x+1} + e^{t \cdot x}\sqrt{x + 1{,}5}.$	t, a, x
12	$y = \dfrac{a^2 \cdot x - e^{-x} \cdot \cos(a \cdot x)}{a \cdot x - e^{-x} \cdot \sin(a \cdot x)};$ $z = \ln(a+x) + e^{2 \cdot x} \ln(a^2 + x^2).$	a, x
13	$y = \sin^3(x^2 + a^2) - \sqrt{x/b};$ $z = x^2/a + \cos(x+b)^3.$	a, x, b
14	$y = x^2 \cdot (x+1) - \sin^2(x+a);$ $z = \sqrt{x \cdot b/a} + \cos^2(x+b)^3.$	a, x, b
15	$z = \dfrac{-t + \sqrt{t^2 + 8 \cdot a^2}}{4 \cdot a};$ $w = (x^2 + y^2 + a \cdot x)^2.$	x, y, t, a

№ вар.	Расчетные формулы	Входные переменные
16	$z = c^2 \cdot \cos(x) + \sqrt{c^4 \cdot \cos^2(2 \cdot x)}$; $w = (x^2 + y^2)^2 - 2 \cdot c^2 \cdot (x^2 - y^2)$.	x, y, c
17	$z = \frac{4 \cdot a \cdot (x + a)}{2 \cdot a + x} + \sin\left(\frac{x}{2 \cdot a}\right)$; $w = \pi \cdot a^2 \cdot \frac{3 \cdot x + 2 \cdot a}{x}$	x, a
18	$y = (a + x)^3 \cdot \cos(x) + \ln(\cos(x))$; $z = x^{2/3} + e^{(a-x)^2}$.	x, a
19	$y = \frac{ae^{x/a} + e^{-x/a}}{2}$; $z = \cos\left(a + \frac{x^2}{2 \cdot a}\right)^{3/2}$.	x, a
20	$y = \frac{a \cdot x \cdot \sqrt{b \cdot x} + a \cdot x^2}{(b - x)^2}$; $z = (a + b) \cdot \sin(x) + b \cdot a \cdot \sin^2\left(\frac{b + a}{x}\right)$.	x, a, b

Порядок выполнения работы

1. Ознакомиться с возможностями среды Microsoft Visual Studio C++.

2. Изучить:

- работу с переменными вещественного типа;
- стандартные математические функции библиотеки <**math.h**>;
- правила записи арифметических выражений;
- организацию консольного ввода и вывода данных.

3. Разработать алгоритм решения в соответствии с заданием.

4. Составить программу решения задачи.

5. Подготовить тестовый вариант исходных данных и вычислить для них вручную или с помощью калькулятора значения вычисляемых в программе величин.

6. Набрать программу в редакторе системы программирования Microsoft Visual Studio C++.

7. Выполнить программу и проверить правильность работы программы на тестовом варианте исходных данных. При наличии ошибок ввести исправления в программу.

8. Модифицировать программу таким образом, чтобы вывод вычисленных значений переменных осуществлялся в соответствии со следующей разметкой строк:

```
*************************
* Результаты вычислений *
*************************
 <пропуск одной строки>
   <имя переменной> = ...
   <имя переменной> = ...
 <пропуск одной строки>
*************************
```

9. Оформить отчет к работе. Отчет должен содержать:
- цель работы;
- задание к работе (с указанием варианта);
- описание алгоритма;
- исходный код программы на языке Си/Си++;
- тестовое задание (с аналитическим расчетом результатов);
- результаты тестирования (в том числе скриншот работы программы);
- выводы.

Контрольные вопросы и задания

1. Как в программе вводятся в употребление переменные?
2. Укажите старшинство выполнения операций при вычислении арифметического выражения.
3. Приведите примеры стандартных математических функций библиотеки **math.h**. Какие из этих функций использованы в вашей программе?
4. Укажите функции языка Си/Си++ для ввода и вывода данных.
5. Укажите, каким образом в языке Си/Си++ осуществляется управление размещением данных на строке и по строкам.

Глава 3. РЕАЛИЗАЦИЯ ВЕТВЯЩИХСЯ И ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

3.1. Реализация ветвящихся алгоритмов

Ветвление является одной из трех основополагающих управляющих конструкций структурного программирования (наряду с последовательностью и циклом).

В качестве примера на рисунке 3.1 приведена блок-схема алгоритма, в котором используется ветвление.

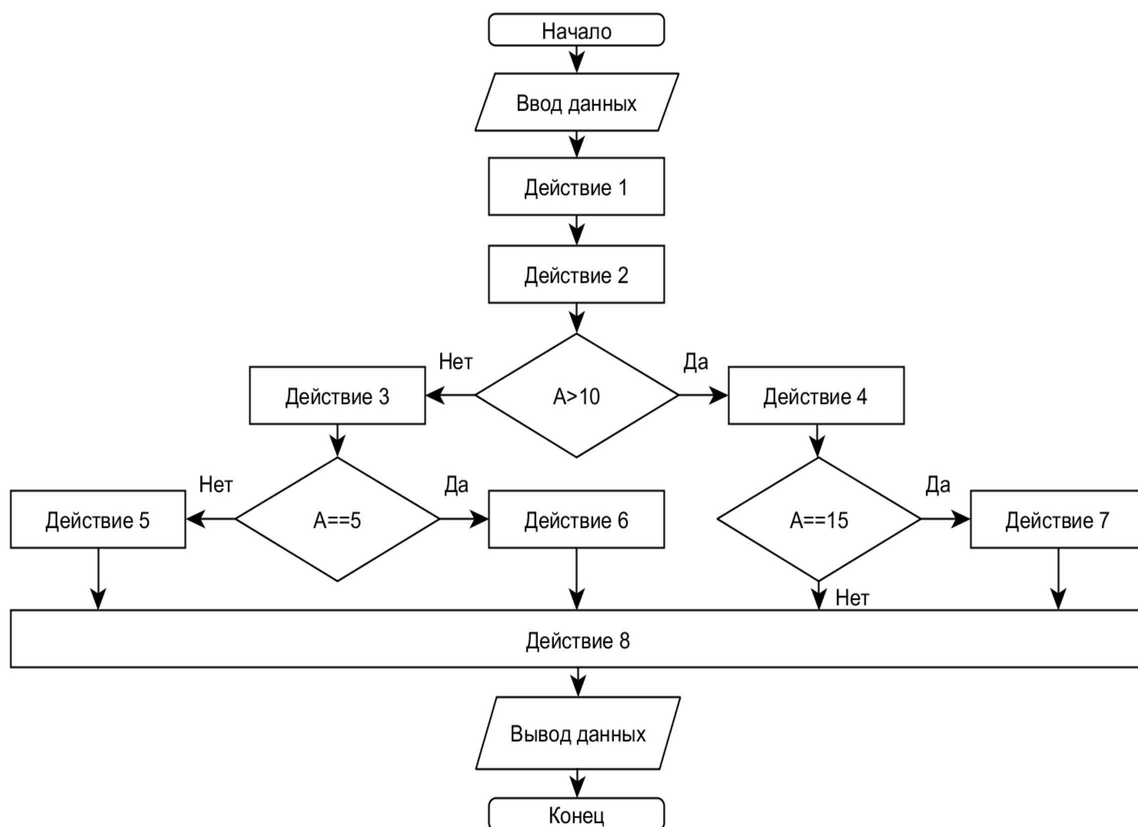

Рисунок 3.1 – Блок-схема алгоритма, использующего ветвление

Рассмотрим возможные сценарии работы алгоритма, представленного на рисунке 3.1:

- Если A = 5, то: Ввод данных → Действие1 → Действие 2 → Действие 3 → Действие 6 → Действие 8 → Вывод данных;
- Если A = 7, то: Ввод данных → Действие 1 → Действие 2 → Действие 3 → Действие 5 → Действие 8 → Вывод данных;

- Если A = 15, то: Ввод данных→ Действие 1 → Действие 2 → Действие 4 → Действие 7 → Действие 8 → Вывод данных;
- Если A = 20, то: Ввод данных → Действие 1 → Действие 2 → Действие 4 → Действие 8 → Вывод данных.

*Оператор **if** (условный оператор)*

Наиболее простым способом реализации ветвления в Си/Си++ является использование оператора **if** («если» в переводе с английского). Он записывается следующим образом:

if (проверка_условия) оператор_1; **else** оператор_2;

Если условие в скобках принимает истинное значение, выполняется оператор_1, если ложное – оператор_2. В операторе **if** слово **else** может отсутствовать, что является сокращенной формой оператора **if**.

В качестве оператора_1 или оператор_2 можно использовать любой оператор языка Си/Си++. Если хотя бы один из них является оператором **if**, его называют вложенным. Согласно принятому в языке Си/Си++ соглашению слово **else** всегда относится к ближайшему предшествующему ему **if**.

Вернемся к рисунку 3.1 и внимательно рассмотрим условие «A==15». Его можно представить в словесной форме в следующем виде: «Если A = 15, то выполнить действие 7, в противном случае продолжить дальнейшее выполнение программы».

В предлагаемой конструкции имеются следующие составные части:

- слово «если» (на Си/Си++ это будет **if**);
- условие A = 15 (на Си/Си++ это будет (A == 15));
- выполнить: «действие 7»;
- дальнейшие действия: «действие 8» и т.д.

Для реализации этих действий потребуется оператор **if** сокращенной формы, т.к. в нем отсутствует ключевое слово **else**.

В таблице 3.1 приводится код, реализующий указанные действия.

Другим примером является условие A==5. Его можно представить в словесной форме в следующем виде: «Если A = 5, то выполнить условие 6, иначе выполнить условие 5, затем продолжить дальнейшее выполнение программы».

Таблица 3.1. Пример использования оператора if

Код	Комментарии
int A = 15;	// объявление // переменой
if (A == 15)	// условие
operation_7();	// действие
operation_8();	// последовательность // дальнейших действий

В предлагаемой конструкции имеются следующие составные части:

- слово «если» (на Си/Си++ это будет **if**);
- условие A = 5 (на Си/Си++ это будет (A == 5));
- выполнить «действие 6»;
- слово «иначе» (на Си/Си++ это будет **else**);
- выполнить «действие 5»;
- дальнейшие действия: «действие 8» и т.д.

В таблице 3.2 приводится код, реализующий указанные действия.

Таблица 3.2. Пример использования ключевого слова else

Код	Комментарии
int A = 5;	// объявление // переменной
if (A == 5)	// условие
operation_6();	// действие если условие // истинно
else operation_5();	// действие если условие // ложно
operation_8();	// последовательность // дальнейших действий

Следует отметить, что условие A == 5 является логическим, таким образом, если условие истинно, то выполняется первое действие, а если ложно, то второе.

Если, при истинности или ложности условия требуется выполнить не одно, а несколько действий, то все эти операции, объединяются в единый «блок» и обрамляются фигурными скобками **{ }**.

В таблице 3.3 приведена программа, реализующая алгоритм, представленный на рисунке 3.1.

Таблица 3.3. Пример вложенного ветвления

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение библиотек
#define operation_1() puts("1") #define operation_2() puts("2") #define operation_3() puts("3") #define operation_4() puts("4") #define operation_5() puts("5") #define operation_6() puts("6") #define operation_7() puts("7") #define operation_8() puts("8")	// директивы, имитирующие // работу «действий»
int main(void) {	// основная функция
int A;	// объявление переменных
scanf("%d", &A);	// ввод данных
operation_1(); operation_2();	// действия 1 // действие 2
if (A > 10) {	// условие A > 10 // начало «блока»

operation_4();
if (A == 15)
operation_7();
}
	// вложенное условие

else
{
operation_3();
if (A == 5)
operation_6();
else
operation_5();
}
	// второе вложенное условие

operation_8();	// дальнейшие действия
printf("Exit\n");	// вывод данных
return 0; }	// конец программы

Оператор ***switch / case***

Оператор **if** позволяет осуществить выбор только между двумя вариантами. Однако существуют ситуации, когда требуется произве-

сти выбор из трех и более вариантов. Для реализации таких случаев, в Си/Си++ предусмотрен оператор **switch / case**.

Он записывается в следующем формальном виде:

```
switch (целочисленное выражение)
{
  case константа_1:  операторы_1; break;
  case константа_2:  операторы_2; break;
  ........              ........          ........
  default:         операторы_default;
}
```

Оператор **switch** выполняется следующим образом:

- вычисляется целочисленное выражение в скобках оператора **switch**;
- полученное значение сравнивается с метками (константами) в опциях **case**, сравнение производится до тех пор, пока не будет найдена метка, соответствующая вычисленному значению целочисленного выражения;
- выполняется оператор соответствующей метки **case**;
- если соответствующая метка не найдена, то выполнится оператор в опции **default**.

Для прекращения последующих проверок после успешного выбора некоторого варианта используется оператор **break**, обеспечивающий немедленный выход из переключателя **switch**.

Допускаются вложенные конструкции **switch**.

Рассмотрим простой пример алгоритма, приведенный на рисунке 3.2. В данном примере требуется выполнить разделение программы одновременно на 3 ветви, в зависимости от значения переменной А.

Реализовать такой алгоритм можно, как и ранее, с применением условия **if**, следует даже отметить, что в некоторых случаях такое решение будет являться единственно верным, т.к. существует лишь один допустимый сценарий использований оператора **switch / case**. Рассмотрим этот сценарий:

- в условии используется целочисленная переменная (например, **int**);

• каждая ветвь связана с конкретным значением переменной из условия, иными словами – условие также является **целочисленным** (а не логическим как в случае с **if**).

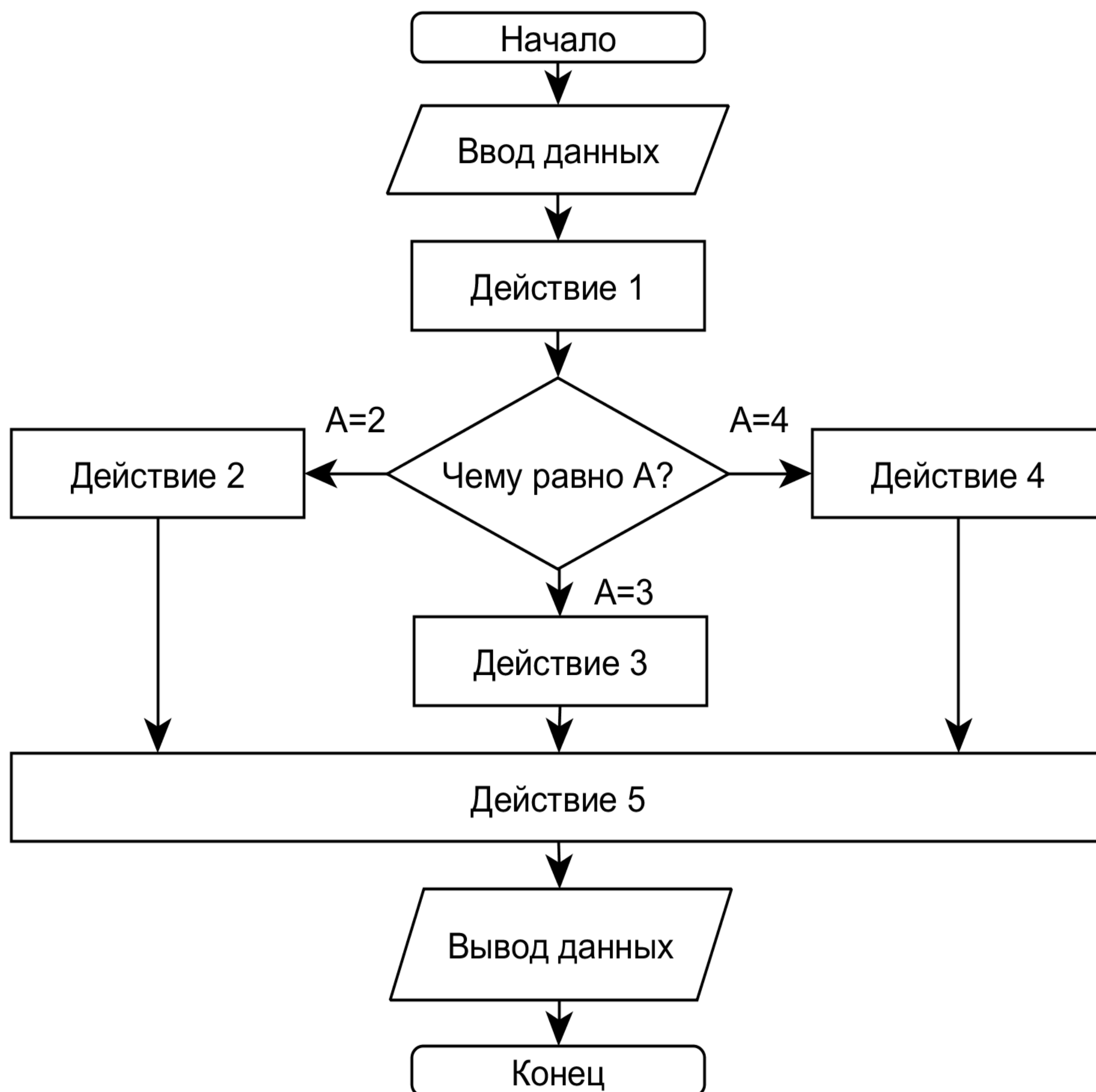

Рисунок 3.2 – Блок-схема алгоритма, использующего оператор switch / case

В таблице 3.4 приведен пример программы, составленной по алгоритму из блок-схемы (рисунок 3.2).

Таблица 3.4. Пример использования конструкции switch / case

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение библиотек
#define operation_1() puts("1") #define operation_2() puts("2") #define operation_3() puts("3") #define operation_4() puts("4") #define operation_5() puts("5")	// директивы, имитирующие // работу «действий»
int main(void) {	// основная функция
int A;	// объявление переменных
scanf("%d", &A);	// ввод данных
operation_1();	// действия 1

switch (A)
{
case 2: operation_2(); break;
case 3: operation_3(); break;
case 4: operation_4(); break;
}
	// операция ветвления

operation_5();	// дальнейшие действия
printf("Exit\n");	// вывод данных
return 0; }	// конец программы

Следует обратить внимание, что в операторе **switch / case** допустимо писать последовательность операций в строку, а не в столбик.

Оператор **switch /case**, имеет свой аналог условия **else**. Оно, как отмечалось выше, называется **default** и означает следующее: «если ни одно из вышеуказанных условий неверно, то».

В таблице 3.5 приведена программа, в которой используется ключевое слово **default**.

Таблица 3.5. Пример использования ключевого слова default

operation_1();	// действия 1

switch (A)
{
case 2: operation_2(); break;
case 3: operation_3(); break;
default: default_operation(); break;
}
	// операция ветвления

3.2. Реализация циклических алгоритмов

Цикл – разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.

Операция цикла часто используется в математике для расчета, например арифметических и геометрических последовательностей, интегралов и других операций. Рассмотрим простой пример задачи, решение которой предполагает использование циклических операций:

$$ans = \sum_{i=1}^{5} \frac{value + i}{5}.$$

Данный пример можно решать двумя способами, первый из них – прямое решение:

$$\sum_{i=1}^{5} \frac{value + i}{5} = \frac{value + 1}{5} + \frac{value + 2}{5} + \frac{value + 3}{5} + \frac{value + 4}{5} + \frac{value + 5}{5} =$$

$$= \frac{5 \cdot value + (1 + 2 + 3 + 4 + 5)}{5} = \frac{5 \cdot value + 15}{5} = value + 3.$$

Однако возникает вопрос, а что если пример будет значительно сложнее, а количество итераций (5 в данном примере) станет значительно больше? Одно из решений состоит в использовании цикла.

Для реализации циклических алгоритмов в языках программирования используются операторы цикла. В языке Си/Си++ имеются операторы цикла **for**, **while** и **do while**.

Оператор цикла ***for*** *(оператор цикла со счетчиком)*

Первый из них формально записывается, в следующем виде:

for (выражение_1; выражение_2; выражение_3) тело_цикла

Тело цикла составляет либо один оператор, либо несколько операторов, заключенных в фигурные скобки { ... } (после блока, точка с запятой не ставится). В выражениях 1, 2, 3 фигурирует специальная переменная, называемая переменная счетчика (иттератор). По ее значению устанавливается необходимость повторения цикла или выхода из него.

Выражение_1 присваивает начальное значение переменной, выражение_3 изменяет его на каждом шаге счетчика, а выражение_2 проверяет, не достигло ли оно граничного значения, устанавливающего необходимость выхода из цикла.

Для решения предлагаемого примера, воспользуемся циклом со счетчиком **for**.

Блок-схема алгоритма решения указанного выше примера с использованием цикла со счетчиком приведена на рисунке 3.3.

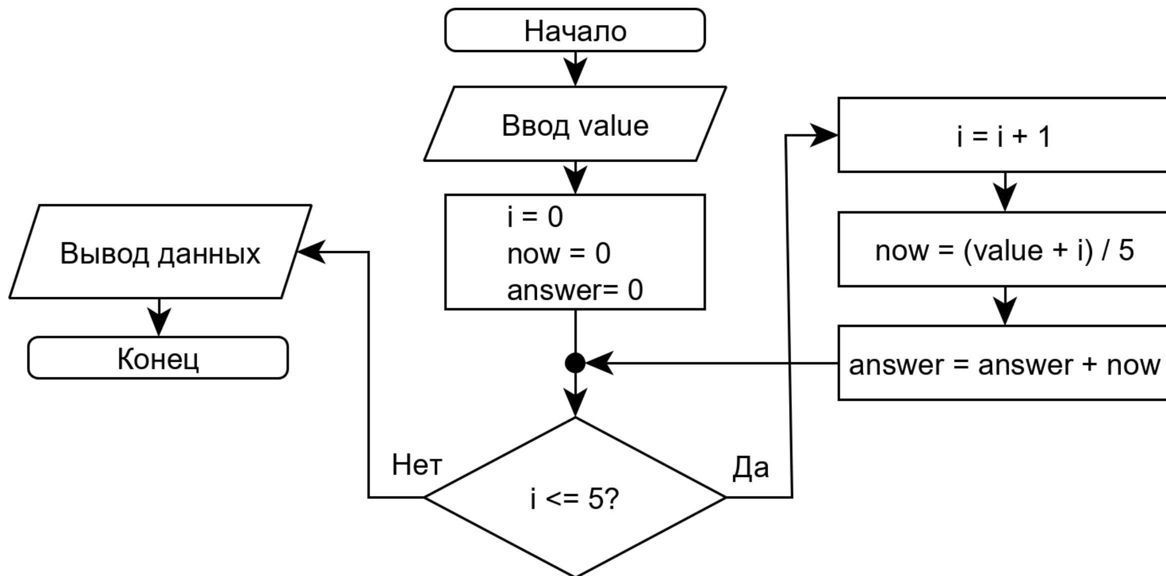

Рисунок 3.3 – Блок-схема алгоритма использующего цикл со счетчиком

При реализации цикла со счетчиком, в качестве переменной счетчика можно использовать переменную любого типа (чаще всего – **int**). Эта переменная называется итератором и обычно имеет называние i (или j).

При использовании оператора **for** требуется:

- выполнить присвоение начального значения счетчика (в примере i = 0);
- указать условие, при котором цикл продолжается (в примере i<=5);
- задать инструкцию изменения (увеличения или уменьшения) переменной-счетчика, выполняемую на каждой итерации (при единичном шаге используется операция **i++** – инкремент).

Следует обратить особое внимание на инструкцию изменения переменной счетчика. В качестве этой инструкции чаще всего используется операция инкремента счетной переменной (если необходимо увеличивать значение счетчика) или декремента счетной переменной (если требуется уменьшать значение счетчика). Уменьшать или увеличивать счетчик допускается и другими способами но, как правило, используются только операции инкремента и декремента.

В таблице 3.6 приведена программа, реализующая алгоритм, обозначенный на рисунке 3.3.

Таблица 3.6. Пример использования ключевого слова for

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение библиотек
int main(void) {	// основная функция
int i, value; float now, ans = 0;	// объявление переменных
scanf("%d", &value);	// ввод данных
for(i=1; i<=5; i++) {	// цикл со счетчиком
now = (value + i) / 5.0; ans = ans + now;	// обратите внимание на // тип переменной ans
}	// завершение блока цикла
printf("Answer = %.2f\n", ans);	// вывод данных
getchar();	// пауза
return 0; }	// конец программы

*Оператор цикла **while** (оператор цикла с предусловием)*

Рассмотрим следующий пример: необходимо определить сумму квадратов чисел лежащих в диапазоне от нуля до десяти с шагом 0,17.

Решение такой задачи возможно с применением цикла с предусловием. Для реализации такого цикла в языке Си/Си++ существует оператор **while**.

Его общая форма записи

While (Условие)
{
    Блок Операций;
}

Суть работы цикла **while** состоит в следующем:

- пока условие истинно, выполняется блок цикла
- если при проверке условия оказалось, что оно ложно, цикл заканчивается, и блок не выполняется.

Пример реализации цикла с предусловием для решения нашей задачи приведен в таблице 3.7.

Таблица 3.7. Пример использования ключевого слова while

Код	Комментарии
float ans = 0; float now = 0;	// переменная с ответом // переменная счетчика
while (now <= 10.0) {	// обратите внимание, именно меньше // или равно!
ans = ans + now*now;	// сумма квадратов
now = now + 0.17;	// ручное увеличение счетчика
}	// конец блока

В таблице 3.8 представлены значения переменной now на всех итерациях, причем именно в начале итераций, в строке ans = ans + now*now.

Таблица 3.8. Описание итерационного цикла

Итерация	1	2	3	4	5	6	7	…	59
now	0,00	0,17	0,34	0,51	0,68	0,85	1,02		9,86
*now*now*	0,00	0,03	0,12	0,26	0,46	0,72	1,04		97,22

В указанном ряду представлены именно те числа, которые мы бы и хотели получить при решении поставленной задачи – это числа от 0 и до наиболее близкого к 10 – числа 9,86 (т.к. следующим будет 10,03). Если мы попытаемся поменять местами строки в блоке **while** местами, то сразу получим неверный ответ! Поэтому, следует запомнить правило: в цикле с предусловием увеличение счетчика следует производить после всех остальных операций.

Важной разновидностью цикла является такой цикл, который повторяется бесконечное количество раз. Для цикла **while** – это цикл, условие которого всегда истинно. Аналогичная конструкция имеется и для цикла **for**. В таблице 3.9 приведен пример бесконечного цикла.

Таблица 3.9. Пример бесконечного цикла

Код	Комментарии
bool led = true;	// объявление переменных

while (true)
{
  led = !led;
  delay();
}
	// бесконечный цикл, который
	// мигает светодиодом

for(;;)
{
  printf("+");
}
	// другой пример бесконечного цикла
	// который выводит в консоль «+»

Следует отметить следующие обстоятельства:

- Если вышеуказанную программу воплотить в жизнь, то она никогда не дойдет до строки **for(;;)**.
- Бесконечный цикл широко применяется в микропроцессорной технике, а также при реализации многозадачности.
- Бесконечный цикл можно завершить с помощью специальной операции выхода – ключевого слова **break**.

Оператор цикла ***do...while*** *(оператор цикла с постусловием)*

Общая форма записи оператора:

do {
  БлокОпераций;
} **while** (Условие);

Цикл **do...while** — это цикл с постусловием, где истинность выражения, проверяющего **Условие** проверяется после выполнения **Блока Операций**, заключенного в фигурные скобки. Тело цикла выполняется до тех пор, пока выражение, проверяющее **Условие**, не станет ложным, то есть тело цикла с постусловием выполнится хотя бы один раз.

Использовать цикл **do...while** лучше в тех случаях, когда должна быть выполнена хотя бы одна итерация, либо когда инициализация объектов, участвующих в проверке условия, происходит внутри тела цикла.

При реализации циклических алгоритмов часто приходится использовать операторы **break** и **continue.**

Оператор разрыва **break** прерывает выполнение операторов **for, while**, **do while** и **switch**. Он может содержаться только в теле этих операторов. Управление передается оператору программы, следующему за прерванным оператором.

Оператор продолжения **continue** передает управление на следующую итерацию в операторах цикла **for, while**, **do while**. Он может содержаться только в теле этих операторов. В операторе **while** следующая итерация начинается с вычисления условного выражения. В операторе **for** следующая итерация начинается с вычисления выражения приращения, а затем происходит вычисление условного выражения.

Рассмотрим пример: при выполнении цикла происходит попытка деления на ноль. В таком случае необходимо срочно завершить выполнение цикла и сообщить пользователю об ошибке. Сделать это можно с помощью оператора **break**. Пример использования оператора **break** приведен в таблице 3.10.

Таблица 3.10. Пример использования оператора break

Код	Комментарии
float ans = 0; int i;	// объявление переменных
for(i=1; i<=100; i++) {	

if ((i-50) == 0)
{
printf("Fatal Error!\n");
break;
}
	// условие выхода

ans = ans + 1.0 / (i-50); }	// потенциально опасное // выражение

Оператор **break** позволяет выйти из цикла в любой момент времени, при этом следует помнить два обстоятельства:

- Если цикл является вложенным (цикл в цикле), то с помощью **break** вы сможете выйти только из вложенного цикла, оставаясь при этом в основном.
- Если во время выполнения цикла вы вызывали функцию, то находясь в самой функции, вы никак не сможете использовать команду **break** для выходя из цикла.

Другую дополнительную возможность, связанную с циклами реализует оператор **continue**, который останавливает текущую итерацию цикла и сразу же начинает следующую.

В таблице 3.11 приводится пример использования ключевого слова continue.

Таблица 3.11. Пример использования ключевого слова continue

Код	Комментарии

int first = 0;
int second = 0;
int third = 0;
int i;
	// объявление переменных

for (i=1; i<=10; i++) {	// цикл // начало блока цикла
first = first + 2;	// первая операция блока

if (i == 3)
{
continue;
}
	// условие выполнения
	// операции continue

second = second + 2; third = third + 2;	// вторая операция блока // третья операция блока
}	// конец блока цикла

Цикл наряду с последовательностью и ветвлением является основой структурного программирования.

Вопросы для самопроверки

1. Что такое ветвление?
2. Приведите пример, использующий ключевое слово **if**.
3. Приведите пример, использующий ключевое слово **else**.
4. Приведите пример, использующий ключевое слово **switch**.
5. Объясните назначение ключевого слова **default**.
6. Что такое цикл?
7. Приведите пример, использующий ключевое слово **for**.
8. Приведите пример, использующий ключевое слово **while**.
9. В чем состоит отличие ключевых слов **break** и **continue**?
10. Для чего используется бесконечный цикл?

Практические задания

1. Необходимо написать программу на языке Си для решения следующей задачи. Даны два числа целого типа: first и second. Если их значения не равны, то присвоить каждой переменной большее из этих значений, а если равны, то присвоить переменным нулевые значения. Вывести новые значения переменных first и second.

2. Необходимо написать программу на языке Си для решения следующей задачи. Даны три числа. Необходимо найти среднее из них (то есть число, расположенное между наименьшим и наибольшим).

3. Необходимо написать программу на языке Си для решения следующей задачи. Прямоугольный треугольник имеет катет длиной «first_cat» и гипотенузу длиной «hypotenuse». Необходимо определить длину второго катета «second_cat» и площадь треугольника «area». Дополнительно необходимо определить недопустимые условия для решения задачи и преодолеть эту проблему

4. Необходимо написать программу на языке Си для решения следующей задачи. Известны значения двух переменных: «first» и «second». Необходимо определить площадь прямоугольника, круга и треугольника, представленных на рисунке 3.4. Дополнительно необходимо определить какая из площадей является наибольшей.

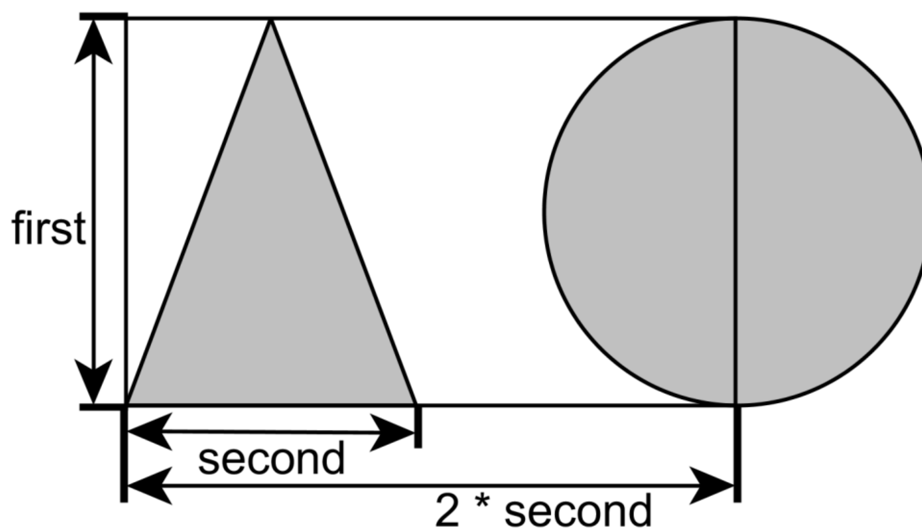

Рисунок 3.4 – Геометрическое представление задачи № 4

5. Необходимо написать программу на языке Си для решения следующей задачи. Дано квадратное уравнение $a \cdot x^2 + b \cdot x + c = 0$. Известно значение коэффициентов *a, b* и *c*. Необходимо определить корни уравнения *x*1 и *x*2. Дополнительно необходимо определить недопустимые условия для решения задачи и преодолеть эту проблему.

Практическая работа № 2. **Программирование алгоритмов разветвляющейся и циклической структуры**

Цель работы – овладение практическими навыками программирования вычислительного процесса разветвляющейся и циклической структуры.

Задание к работе

В соответствии с вариантом задания (таблицу 3.12), в среде Visual Studio необходимо разработать программу выполняющую:

- расчет значений аргумента x и функции $y(x)$ (формулы расчета, диапазон изменения аргумента и его шаг указаны таблице 3.12);
- вывод в консоль исходных данных (переменная x) и результатов расчета (переменная y) с указанием имен переменных в виде таблицы:

```
              Таблица функции y(x)
-------------------------------------
|        X        |         Y       |
-------------------------------------
|   <значение>    |   <значение>    |
|   <значение>    |   <значение>    |
|      ...        |      ...        |
-------------------------------------
```

Например, если $y(x)=2\cdot x$, $x\in[1..10]$, $h=2$, то результат работы программы будет следующим:

```
           Таблица функции y(x) = 2*x
-------------------------------------
|        X        |         Y       |
-------------------------------------
|        1        |         2       |
|        3        |         6       |
|        5        |        10       |
|        7        |        14       |
-------------------------------------
```

Таблица 3.12. Варианты заданий к практической работе № 2

№ вар.	Расчетные формулы	Исходные данные	Диапазон изменения аргумента	Шаг изм. арг. (h)
1	$y=\begin{cases} a\cdot t^2\cdot\ln(t) & при\quad 1<t<2; \\ 1 & при\quad t\le 1; \\ e^{a\cdot t}\cdot\cos(b\cdot t) & при\quad t\ge 2. \end{cases}$	$a=-0{,}5$, $b=2$	$0{,}1\le t\le 3$	0,15
2	$y=\begin{cases} (2\cdot x+1)\cdot\sin(x+1) & при\quad x=0; \\ x^2\cdot\sin(x) & при\quad x>0; \\ \sqrt{x}\cdot\sin(x^a) & при\quad x<0. \end{cases}$	$a=4$	$-0{,}5\le x\le +2$	0,1
3	$y=\begin{cases} \pi\cdot x^2-7/x^2 & при\quad x<1{,}3; \\ a\cdot x^3+7\cdot\sqrt{7} & при\quad x=1{,}3; \\ \lg(x+7\cdot\sqrt{7} & при\quad x>1{,}3. \end{cases}$	$a=1{,}5$	$0{,}8\le x\le 2$	0,1
4	$y=\begin{cases} \dfrac{\sqrt{a+x^2}}{\sin(x)} & при\quad x^2+1\ge 2; \\ b\cdot\ln\lvert x\rvert-tg(x^2+1) & при\quad x^2+1<2. \end{cases}$	$a=1$, $b=0{,}35$	$-0{,}3\le x\le 1{,}8$	0,15
5	$y=\begin{cases} \log_2(a\cdot x) & при\quad x-1\ge 0; \\ \ln(\arcsin(bx)) & при\quad x-1<0. \end{cases}$	$a=5$, $b=2$	$0{,}1\le x\le 3$	0,15
6	$y=\begin{cases} a\cdot x^2+b\cdot x+c & при\quad x<1{,}2; \\ a/x+\sqrt{x^2+1} & при\quad x=1{,}2; \\ (a+b\cdot x)\cdot\sqrt{x^2+1} & при\quad x>1{,}2. \end{cases}$	$a=2{,}8$, $b=-0{,}3$, $c=4$	$1\le t\le 2$	0,05
7	$y=\begin{cases} (e^{a\cdot x}+b)/\sqrt{x} & при\quad x<5; \\ \cos(x^2) & при\quad x=5; \\ \log_5(x) & при\quad x>5. \end{cases}$	$a=2$, $b=4{,}5$	$3\le x\le 7$	0,15
8	$y=\begin{cases} 1{,}5\cdot\cos^2(x) & при\quad x<1; \\ 1{,}8\cdot a\cdot x & при\quad x=1; \\ (x-2)^2+6 & при\quad 1<x<2; \\ 3\cdot tg(x) & при\quad x\ge 2. \end{cases}$	$a=2{,}3$	$0{,}2\le x\le 2{,}8$	0,2

№ вар.	Расчетные формулы	Исход-ные дан-ные	Диапазон изменения аргумента	Шаг изм. арг. (h)
9	$y=\begin{cases}\sqrt{x^2-b}/\ln(x) & при \quad x^2-b>0;\\ \sin(x) & при \quad x^2-b<0;\\ e^{2\cdot x+b} & при \quad x^2-b=0.\end{cases}$	$b=1$	$0{,}5\le x\le 2$	0,1
10	$y=\begin{cases}x^3\sqrt{x-a} & при \quad x>a;\\ x\cdot\sin(a\cdot x) & при \quad x=a;\\ e^{-a\cdot x}\cos(a\cdot x) & при \quad x<a.\end{cases}$	$a=2{,}5$	$1\le x\le 5$	0,5
11	$y=\begin{cases}\log_3(a\cdot x)+b^x & при \quad x\ge 2{,}1;\\ \dfrac{\ln\lvert\sin(x)\rvert}{\sqrt{\lvert b/3-a/2\cdot\sin(x)\rvert}} & при \quad x<2{,}1.\end{cases}$	$a=4$, $b=1{,}5$	$-0{,}5\le x\le 3$	0,2
12	$y=\begin{cases}b\cdot x-\lg(b\cdot x) & при \quad b\cdot x<1;\\ 1 & при \quad b\cdot x=1;\\ b\cdot x+\lg(b\cdot x) & при \quad b\cdot x>1.\end{cases}$	$b=1{,}5$	$0{,}1\le x\le 2$	0,1
13	$y=\begin{cases}\lg^a(x) & при \quad x\ge 1;\\ \sin(\ln(x/a)) & при \quad 0<x<1;\\ x+17 & при \quad x\le 0.\end{cases}$	$a=2$	$-3\le x\le 3$	0,5
14	$y=\begin{cases}\sqrt{x}+a & при \quad x<0{,}5;\\ \sqrt{x+a} & при \quad x=0{,}5;\\ \cos(x)+a\cdot\sin^2(x) & при \quad x>0{,}5.\end{cases}$	$a=2{,}2$	$0{,}2\le x\le 2$	0,1
15	$y=\begin{cases}3\cdot a\cdot\sin\sqrt{\lvert x\rvert^3} & при \quad x<-5{,}1;\\ 2 & при \quad -5{,}1\le x\le 0;\\ \log_{3\cdot a}(x) & при \quad 0<x<3{,}4;\\ 125 & при \quad x\ge 3{,}4.\end{cases}$	$a=0{,}5$	$-6<x<4$	0,5
16	$y=\begin{cases}\lg(x+1) & при \quad x\ge 1;\\ \sin^2\sqrt{\lvert a\cdot x\rvert} & при \quad x<1.\end{cases}$	$a=20{,}3$	$0{,}5<x<2$	0,1

№ вар.	Расчетные формулы	Исходные данные	Диапазон изменения аргумента	Шаг изм. арг. (h)
17	$y=\begin{cases} \lg^3(x)+x^a & при \quad x<0{,}5; \\ \sqrt{x+a}+1/x & при \quad x=0{,}5; \\ \sin^a(x)+\cos(x) & при \quad x>0{,}5. \end{cases}$	$a=2$	$0{,}1<x<1{,}5$	0,1
18	$y=\begin{cases} e^x+\cos(a\cdot x) & при \quad x<3; \\ (a+b)/(x+1) & при \quad x=3; \\ e^x+\sin(b\cdot x) & при \quad x>3. \end{cases}$	$a=3{,}1$, $b=0{,}1$	$0<x<5$	0,5
19	$y=\begin{cases} a\cdot\lg(x) & при \quad x\geq 1; \\ 2\cdot a\cdot\cos(x)+3\cdot x^2 & при \quad x<1. \end{cases}$	$a=0{,}9$	$0{,}8<x<2$	0,1
20	$y=\begin{cases} a/x+b\cdot x^2 & при \quad x<4; \\ \lvert x\rvert^3 & при \quad x=4; \\ a\cdot x+b\cdot x^2 & при \quad x>4. \end{cases}$	$a=2$, $b=-3$	$2<x<6$	0,3

Порядок выполнения работы

1. Изучить возможности языка Си/Си++ для реализации вычислительного процесса разветвляющейся и циклической структуры.

2. Разработать алгоритм вычисления значения функции в соответствии с заданием.

3. Разработать алгоритм и программу циклического вывода в консоль аргумента функции (переменная x) и значения функции (переменная y) с шагом аргумента h. Значение аргумента должно изменяться в пределах диапазона, заданного в таблице 3.12.

4. Подготовить тесты для проверки правильности функционирования программ, причем число тестов должно соответствовать числу ветвей вычислительного процесса.

5. Набрать программу в редакторе системы программирования Microsoft Visual Studio C++.

6. Протестировать разработанную программу. При наличии ошибок внести изменения в программу.

7. Оформить отчет к работе. Отчет должен содержать:

- цель работы;

- задание к работе (с указанием варианта);
- описание алгоритма;
- исходный код программы на языке Си/Си++;
- тестовое задание (с аналитическим расчетом результатов);
- результаты тестирования (в том числе скриншот работы программы);
- выводы.

Контрольные вопросы и задания

1. Как организовать разветвление вычислений: а) на две ветви; б) на три ветви?
2. Перечислите действия, реализуемые при выполнении условного оператора (**if**) и оператора варианта (**case**).
3. Требуется ли при отладке программы, тестировать все ветви алгоритма?
4. Укажите правила организации цикла.
5. Перечислите операторы цикла языка Си/Си++.

3.3. Рекуррентные вычисления

Рекуррентная формула (от лат. recurrens, родительный падеж recurrentis – возвращающийся), формула, сводящая вычисление i-го члена какой-либо последовательности (чаще всего числовой) к вычислению нескольких предыдущих ее членов.

Примером рекуррентных последовательностей являются:

- Арифметическая прогрессия: числовая последовательность, в которой каждое число, начиная со второго, получается из предыдущего добавлением к нему постоянного числа d (шага или разности прогрессии):

$$a_n = a_{n-1} + d.$$

- Геометрическая прогрессия: последовательность чисел $b_1, b_2, b_3, \ldots$ (называемых членами прогрессии), в которой каждое последующее число, начиная со второго, получается из предыдущего умножением его на определённое число q (называемое знаменателем прогрессии):

$$b_n = b_{n-1} \cdot q.$$

- Последовательность чисел Фибоначчи: элементы числовой последовательности, в которой первые два числа равны либо 1 и 1, либо 0 и 1, а каждое последующее число равно сумме двух предыдущих чисел. Более формально, последовательность чисел Фибоначчи задается линейным рекуррентным соотношением:

$$F_1 = 0,\ F_2 = 1,\ F_n = F_{n-1} + F_{n-2}.$$

Рекуррентная формула часто используется при вычислении суммы ряда.

При суммировании ряда необходимо решить следующие задачи:

1. Свести вычисления к простейшим арифметическим операциям.

2. Уменьшить число этих операций и, следовательно, уменьшить время расчета.

В общем виде, сумма конечного ряда имеет вид:

$$s_n(x) = \sum_{i=0}^{n} a_i(x).$$

Например, для суммы $s_n(x) = 3x + 8x^2 + 15x^3 + ... + n \cdot (n+2) \cdot x^n =$

$$= \sum_{i=1}^{n} n \cdot (n+2) \cdot x^n.$$

Суммирование ряда последовательным вычислением слагаемых и добавлением их к сумме сводит вычисления к простейшим арифметическим операциям, то есть первая задача при этом решается. Что касается второй задачи – уменьшения количества этих операций, – то *многократное* перемножение чисел (вычисление степени или факториала x) вряд ли можно считать рациональным. Задачи сокращения количества операций вычислений решает *рекуррентная* формула, позволяющая вычислить значение очередного члена ряда, используя уже найденное значение предыдущего. Рекуррентная формула имеет вид:

$$a_i(x) = a_{i-1}(x) \cdot z(i, x),$$

где $z(i,x)$ – коэффициент рекурсии.

Коэффициент рекурсии определяется по формуле:

$$Z_i(x) = \frac{a_i(x)}{a_{i-1}(x)}. \quad (1)$$

В нашем примере произвольный член ряда определяется формулой:

$$a_i(x) = i \cdot (i+2) \cdot x^i \text{, где } i = 1, 2, \ldots, n. \qquad (2)$$

В формулу общего члена ряда вместо i подставим (i-1), получим

$$a_{i-1}(x) = (i-1)(i+x) \cdot x^{i-1}. \qquad (3)$$

Подставим выражения (2) и (3) в формулу (1) получим формулу для определения коэффициента рекурсии:

$$z(i,x) = \frac{i \cdot (i+2)}{(i-1) \cdot (i+1)} \cdot x.$$

Отсюда рекуррентная формула для вычисления членов ряда имеет вид

$$a_i(x) = a_{i-1}(x) \cdot \frac{i \cdot (i+2)}{(i-1) \cdot (i+1)} \cdot x.$$

Проверим полученную формулу, что поможет избежать ошибок в алгоритме и, возможно, сэкономить время при отладке программы.

Подставив $n = 1$ в формулу общего члена ряда (2), получаем $a_1 = 3x$.

Далее определим по рекуррентной формуле a_2 и a_3, сверяя результаты с соответствующими членами ряда:

при $i = 2$, $a_2 = a_1 z_1(x) = \dfrac{3x \cdot 2 \cdot 4 \cdot x}{1 \cdot 3} = 8x^2$;

при $i = 3$, $a_3 = a_2 z_2(x) = \dfrac{8x^2 \cdot 3 \cdot 5 \cdot x}{2 \cdot 4} = 15x^3$.

Совпадение полученных значений с членами ряда показывает, что коэффициент рекурсии рассчитан правильно.

Вопросы для самопроверки

1. Что такое рекуррентная формула?
2. Для чего она используется в программировании?
3. Как выводится рекуррентная формула при вычислении суммы ряда?

Практическая работа № 3. **Программирование алгоритмов с вложенным циклом**

Цель работы – овладение практическими навыками вывода рекуррентных формул и программирования алгоритмов с вложенными циклами.

Задание к работе

В соответствии с вариантом задания (таблица 3.13), в среде Visual Studio необходимо разработать программу, выполняющую:

- вычисление суммы $s = \sum_{n=1}^{N} a_n(x)$ при изменении аргумента в указанном по заданию диапазоне с шагом $h = (b - a)/9$ (параметры a и b указаны в графе «Диапазон изменение аргумента» в таблицы 3.13);

- вычисление функции $y(x)$ для того же аргумента (x), что и для суммы s.

- вывод в консоль результатов вычисления, а также исходных данных в виде таблицы:

```
Таблица сопоставления суммы s и функции y(x)
    ----------------------------------------
    |  X  |   сумма s   |   функция y(x)   |
    ----------------------------------------
    | ... |     ...     |       ...        |
    | ... |     ...     |       ...        |
    ----------------------------------------
```

Таблица 3.13. Варианты заданий к практической работе № 3

№ вар.	Сумма (s)	Диапазон изменения аргумента: [a..b]	n	Функция y(x)
1	$s = 1 + \frac{x \cdot \ln^2(3)}{1!} + ... + \frac{x^n \cdot \ln^n(3)}{n!}$	$0{,}1 \le x \le 1$	10	$y(x) = 3^x$
2	$s = 1 - \frac{3}{2}x^2 + ... + (-1)^n \frac{2 \cdot n^2 + 1}{(2 \cdot n)!} x^{2n}$	$0{,}1 \le x \le 1$	35	$y(x) = (1 - \frac{x^2}{2})\cos(x) - \frac{x}{2}\sin(x)$
3	$s = x - \frac{x^3}{3!} + ... + (-1)^n \cdot \frac{x^{2 \cdot n + 1}}{(2 \cdot n + 1)!}$	$0{,}1 \le x \le 1$	10	$y(x) = \sin(x)$
4	$s = 1 + \frac{x}{1!} + ... + \frac{x^n}{n!}$	$1 \le x \le 2$	15	$y(x) = e^x$

№ вар.	Сумма (s)	Диапазон изменения аргумента: [a..b]	n	Функция y(x)
5	$s=1-\frac{x^2}{2!}+...(-1)^n\cdot\frac{x^{2\cdot n}}{(2\cdot n)!}$	$0,1\le x\le 1$	10	$y(x)=\cos(x)$
6	$s=-(1+x)^2+...+(-1)^n\frac{(1+x)^{2\cdot n}}{n}$	$-2\le x\le -0.1$	90	$y(x)=\ln\left(\frac{1}{2+2\cdot x+x^2}\right)$
7	$s=x+\frac{x^5}{5}+...+\frac{x^{4\cdot n+1}}{4\cdot n+1}$	$0,1\le x\le 0,8$	30	$y(x)=\frac{1}{4}\cdot\ln\frac{1+x}{1-x}+\frac{1}{2}arctg(x)$
8	$s=1+\frac{\cos(x)}{1!}+...+\frac{\cos(n\cdot x)}{n!}$	$0,1\le x\le 1$	20	$y(x)=e^{\cos(x)}\cdot\cos(\sin(x))$
9	$s=1+3\cdot x^2+...+\frac{2\cdot n+1}{n!}\cdot x^{2\cdot n}$	$0,1\le x\le 1$	10	$y(x)=(1+2\cdot x^2)\cdot e^{x^2}$
10	$s=\frac{x\cdot\cos(\frac{\pi}{3})}{1}+...+\frac{x^n\cdot\cos(n\cdot\frac{\pi}{3})}{n}$	$0,1\le x\le 0,8$	35	$y(x)=-\frac{1}{2}\ln(1-2\cdot x\cdot\cos\frac{\pi}{3}+x^2)$
11	$s=\frac{x-1}{x+1}+...+\frac{1}{2\cdot n+1}\cdot\frac{(x-1)^{2\cdot n+1}}{(x+1)^{2\cdot n+1}}$	$0,2\le x\le 1$	10	$y(x)=\frac{1}{2}\cdot\ln(x)$
12	$s=x+\frac{x^3}{3!}+...+\frac{x^{2\cdot n+1}}{(2\cdot n+1)!}$	$0,1\le x\le 1$	20	$y(x)=\frac{e^x-e^{-x}}{2}$
13	$s=x\cdot\cos(\frac{\pi}{4})+...+x^n\cdot\cos(n\cdot\frac{\pi}{4})$	$0,1\le x\le 0,8$	40	$y(x)=\frac{x\cdot\cos(\frac{\pi}{4})-x^2}{1-2\cdot x\cdot\cos(\frac{\pi}{4})+x^2}$
14	$s=1+\frac{x^2}{2}+...+\frac{x^{2\cdot n}}{(2\cdot n)!}$	$0,1\le x\le 1$	10	$y(x)=\frac{e^x+e^{-x}}{2}$
15	$s=\frac{x^2}{2}+...+(-1)^{n+1}\frac{x^{2\cdot n}}{2\cdot n\cdot(2\cdot n-1)}$	$0,1\le x\le 0,8$	10	$y(x)=x\cdot arctg(x)-\ln\sqrt{1+x^2}$
16	$s=1+\frac{2\cdot x}{1!}+...+\frac{(2\cdot x)^n}{n!}$	$0,1\le x\le 1$	20	$y(x)=e^{2\cdot x}$
17	$s=1+\frac{2\cdot x}{2}+...+\frac{n^2+1}{n!}\cdot(0,5\cdot x)^n$	$0,1\le x\le 1$	30	$y(x)=\left(\frac{x^2}{4}+\frac{x}{2}+1\right)\cdot e^{0,5\cdot x}$
18	$s=x-\frac{x^3}{3}+...+(-1)^n\cdot\frac{x^{2\cdot n+1}}{2\cdot n+1}$	$0,1\le x\le 0,5$	90	$y(x)=arctg(x)$
19	$s=-\frac{(2\cdot x)^2}{2}+...+(-1)^n\cdot\frac{(2\cdot x)^{2\cdot n}}{(2\cdot n)!}$	$0,1\le x\le 1$	15	$y(x)=2\cdot(\cos^2(x)-1)$
20	$s=3\cdot x+...+n\cdot(n+2)\cdot x^n$	$0.1\le x\le 0.8$	90	$y(x)=\frac{x\cdot(3-x)}{(1-x)^3}$

Сумма s является суммой конечного ряда, в который раскладывается функция $y(x)$, и может служить приближенной заменой этой функции, т.е. $s \approx y(x)$. Поэтому при вычислении суммы для сравнения необходимо вычислять и функцию $y(x)$.

При выполнении практической работы, необходимо составить рекуррентное соотношение, с помощью которого вычисляется сумма s.

Как известно вычисление суммы ряда осуществляется в цикле, а так как такие вычисления по условию задания должны проводиться для различных значений аргумента x из заданного диапазона, потребуется организовать конструкцию вложенный цикл.

Конструкция вложенный цикл – это когда в теле одного цикла (внешнего) размещается один или несколько других циклов, называемых вложенными (внутренними). Такая конструкция строится по принципу «матрешки»: каждый внутренний цикл должен полностью помещаться внутрь внешнего.

Правила организации как внешнего, так и внутренних циклов такие же, как и для простого цикла. Однако параметры этих циклов изменяются не одновременно, т.е. при одном значении параметра внешнего цикла, параметр внутреннего цикла принимает по очереди все свои значения. В качестве примера рассмотрим фрагмент программы (таблица 3.14), который содержит конструкцию вложенный цикл: цикл for с параметром i – внешний цикл, а цикл for с параметром j – вложенный (внутренний) цикл.

Таблица 3.14. Пример конструкции вложенный цикл

Код	**Комментарии**
#include "stdafx.h" #include <stdio.h>	// подключение библиотек
int i, j, x;	// объявление переменных

int main(void)
{
  printf("i j x\n");
  for (i=1; i<=2; i++)
  {
    for (j=1; j<=3; j++)
    {
      x = i + j;
      printf("%01d %01d %01d\n", i, j, x);
    }
  }
  getchar();
  fflush(stdin);
  return 0;
}
	// основная функция
	
	// основной цикл
	
	// вложенный цикл
	
	// вывода данных

В результате выполнения этого программного кода будут получены следующие значения (таблица 3.15).

Таблица 3.15. Результаты выполнения программы таблицы 3.13

i	***j***	***x***
1	1	2
1	2	3
1	3	4
2	1	3
2	2	4
2	3	5

Порядок выполнения работы

1. Изучить:
- организацию вложенных циклов;
- возможности языка Си/Си++ для организации вложенных циклов;
- приемы программирования вычисления суммы.

2. Разработать рекуррентное соотношение в соответствии с задачей из таблицы 3.13 (см. столбец «Сумма (s)»).

3. Разработать алгоритм решения задачи вычисления суммы s и функции $y(x)$.

4. Составить программу решения задачи на языке Си/Си++.

5. Набрать программу в редакторе системы программирования Microsoft Visual Studio C++.

6. Протестировать разработанную программу. При наличии ошибок внести изменения в программу. Основным критерием проверки должно стать сравнение суммы s и функции $y(x)$. Указанные поля для одного и того же аргумента x должны совпадать, как минимум, до второго знака после запятой.

7. Оформить отчет к работе. Отчет должен содержать:

- цель работы;
- задание к работе (с указанием варианта);
- рекуррентное соотношение;
- описание алгоритма;
- исходный код программы на языке Си/Си++;
- результаты тестирования программы (в т.ч. скриншот программы);
- выводы.

Контрольные вопросы и задания

1. Что такое рекуррентное соотношение?

2. Что такое вложенный цикл?

3. В какой последовательности выполняются операции в программе, содержащей вложенный цикл? Приведите пример.

4. Укажите отличия в организации цикла с заданным числом повторений и цикла с предусловием.

5. Какие средства (операторы) языка Си/Си++ целесообразно использовать для организации цикла с заданным числом повторений? Приведите пример.

Глава 4. СЛОЖНЫЕ ТИПЫ ДАННЫХ, СТРУКТУРЫ

4.1. Одномерные массивы, определение и объявление

Разные классы решаемых на ЭВМ задач характеризуются и разными структурами данных, что находит отражение и в соответствующих языках программирования.

Сложный тип данных включает в себя одновременно несколько элементов простого типа (типов).

Рассмотрим наиболее часто используемые структуры данных языка Си/Си++.

Необходимость в массивах возникает всякий раз, когда при решении задачи приходится иметь дело с большим, но конечным количеством однотипных упорядоченных данных.

Массив – это тип данных в виде пронумерованного набора величин одинакового типа, обозначаемого одним именем. Элементы массива располагаются в последовательных ячейках памяти, обозначаются именем массива и индексом. Каждое из значений, составляющих массив, называется его *компонентой* (или *элементом* массива).

Доступ к отдельным элементам массива осуществляется с помощью индексации, то есть через ссылку на массив с указанием номера нужного элемента.

Количество индексов элементов массива определяет *размерность* массива. По этом признаку массивы делятся на одномерные (линейные), двумерные, трехмерные и т.д.

В языке Си/Си++ массив может включать в себя элементы любого простого типа. В качестве примера, рассмотрим создание одномерного массива типа **int,** состоящего из пяти элементов (таблица 4.1).

Таблица 4.1. Пример объявления одномерного массива

Код	Комментарии
int value = 0;	// объявление простой // переменной
int empty_array[5];	// объявление пустого // массива
int array[5] = {0, 1, 3, 9, 0};	// объявление массива // и присвоение значений

При объявлении массива необходимо помнить три правила:

- необходимо указывать желаемое количество элемента в массиве после его имени в квадратных скобках;
- массиву можно сразу же присвоить значение в виде перечисления значений всех его элементов обрамленного в фигурные скобки;
- описанный выше способ присвоения значений всему массиву действует только в момент его объявления. В дальнейшем подобное присвоение запрещено.

Массив array рассмотренный в вышеуказанном примере, хранится в памяти компьютера следующим образом (таблица 4.2).

Таблица 4.2. Хранение элементов массива в памяти компьютера

<table>
<tr><td>Тип элемента</td><td colspan="4">int</td><td colspan="4">int</td><td colspan="4">int</td><td colspan="4">int</td><td colspan="4">int</td></tr>
<tr><td>Индекс элемента</td><td colspan="4">0</td><td colspan="4">1</td><td colspan="4">2</td><td colspan="4">3</td><td colspan="4">4</td></tr>
<tr><td>Значение элмента</td><td colspan="4">0</td><td colspan="4">1</td><td colspan="4">3</td><td colspan="4">9</td><td colspan="4">0</td></tr>
<tr><td>Сдвиг адреса памяти</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td></tr>
</table>

Указанный массив занимает в памяти ЭВМ 20 байт, т.к. каждый элемент типа **int** занимает в памяти 4 байта.

В языке программирования Си/Си++ обращение к отдельным элементам массива осуществляется по его индексу. Причем следует помнить, что индексы начинаются с нулевого и заканчиваются значением N-1, где N – количество элементов в массиве. Например, если вы объявляете массив int array[5], то его первым индексом будет «0», а последним «4».

Обращение к элементу массива по индексу осуществляется с применением квадратных скобок (таблица 4.3).

Таблица 4.3. Обращение к элементам массива по индексу

Код	Комментарии
int value;	// объявление переменной
int array[5] = {0, 1, 3, 9, 0};	// объявление массив
value = array[3];	// чтение из ячейки массива
printf("%d", value);	// вывод данных на экран (число 9)

Указанным выше способом можно не только читать элементы массива, но и записывать данные.

Рассмотрим следующий пример: необходимо считать из консоли массив из 5 целых чисел, а затем найти сумму элементов этого массива. Программный код этого примера приведен в таблице 4.4.

Таблица 4.4. Пример ввода и обработки одномерного массива

Код	Комментарии
int sum, value, i;	// объявление переменных
int array[5];	// объявление массив

for (i=0; i<5; i++)
{
  scanf("%d", &value);
  array[i] = value;
}
	// цикл чтения

sum = 0;
for (i=0; i<5; i++)
{
  sum = sum + array[i];
}
	// обработка

printf("%d", sum);	// вывод результата

При работе с массивами следует особенно внимательно относиться к количеству элементов находящихся в массиве. Нежелательные последствия при работе с массивами могут проявиться в двух случаях:

- При обращении к несуществующему элементу массива, например, если вы объявляете массив int array[5] и при этом обращаетесь к элементу с индексом 5, то программа завершится с ошибкой.
- При объявлении массива нужно четко понимать, сколько элементов вам понадобится, например если вы попытаетесь объявить массив из 1млрд элементов типа int, то на хранение такого массива вам потребуется 4Гб оперативной памяти!

Особенно внимательными нужно быть при создании многомерных массивов.

4.2. Алгоритмы типовых действий с одномерными массивами

1. Ввод массива с клавиатуры

Пример реализации алгоритма приведен в таблице 4.5.

Таблица 4.5. Ввод массива с клавиатуры

Код	Комментарии
int i, arr[10];	// объявление переменных

for (i=0; i<10; i++)
{
  scanf("%d", &arr[i]);
}
	// цикл ввода данных
	
	// ввод данных

2. *Вывод массива на экран*

Пример реализации алгоритма приведен в таблице 4.6.

Таблица 4.6. Вывод массива на экран

Код	Комментарии
int i, arr[10];	// объявление переменных

for (i=0; i<10; i++)
{
  printf("%d ", arr[i]);
  //printf("%d\n", arr[i]);
}
	// цикл вывода данных
	
	// (1) – в строку
	// (2) – в столбец (закомментируйте 1)

3. *Суммирование элементов массива*

Пример реализации алгоритма приведен в таблице 4.7.

Таблица 4.7. Суммирование элементов массива

Код	Комментарии
int i, arr[10], s=0;	// объявление переменных

for (i=0; i<10; i++)
{
  s += arr[i];
}
	// цикл обработки данных
	
	// суммирование

4. *Суммирование двух массивов по элементам* (оба массива должны полностью совпадать как по размерности, так и по типам их элементов).

Пример реализации алгоритма приведен в таблице 4.8.

Таблица 4.8. Суммирование двух массивов по элементам

Код	Комментарии
int i, a[10], b[10], c[10];	// объявление переменных

for (i=0; i<10; i++)
{
  c[i] = a[i] + b[i];
}
	// цикл обработки данных
	
	// суммирование

5. *Удаление элемента из массива:*

Требуется удалить элемент с индексом *k* из массива *A* размером *N*. Удалить элемент, расположенный на *k*-м месте в массиве, можно, сдвинув весь "хвост" массива, начиная с $(k + 1)$ -го элемента, на одну позицию влево, т.е. выполняя операцию $a_i = a_{i+1}$, $i = k, k+1, \ldots, N-2$. В таблице 4.9 приводится реализация указанного алгоритма.

Таблица 4.9. Удаление элемента из массива

Код	Комментарии
int i, k=7, arr[10];	// объявление переменных

for (i=k; i<(10-1); i++)
{
  arr[i] = arr[i+1];
}
	// цикл от k до N-1
	
	// сдвиг на один элемент влево

6. *Включение элемента в заданную позицию массива:*

Перед включением элемента в k—ю позицию необходимо раздвинуть массив, т.е. передвинуть "хвост" массива вправо на одну позицию, начиная с (k + 1)-го элемента, выполняя операцию $a_{i+1} = a_i$, $i = N, N\text{-}1 ,... ,k$. Перемещения элементов массива нужно начинать с конца. В противном случае весь "хвост" будет заполнен элементом a_k. Далее k-му элементу присваивается заданное значение B.

Программный код, реализующий алгоритм включение элемента в заданную позицию массива, приведен в таблице 4.10.

Таблица 4.10. Добавление элемента в массив

Код	Комментарии
int i, index = 2, value = 8; int array[7] = {1,7,9,3,2,1,0};	// объявление // переменных

for (i=6; i>index; i--)
  array[i] = array[i-1];
array[index] = value;
	// вставлен будет
	// элемент с индексом
	// 2 и значением 8

В результате получим новый массив $[1 \quad 7 \quad 8 \quad 9 \quad 3 \quad 2 \quad 1]$. Размер массива увеличивается на 1.

7. *Поиск минимального (максимального) элемента в массиве*

Требуется найти минимальный элемент в массиве и его значение поместить в переменную *ans,* а индекс – в переменную *k.* Пример реализации алгоритма приведен в таблице 4.11.

Таблица 4.11. Поиск минимального элемента в массиве

Код	Комментарии
int i, arr[10]; int k=0, ans=arr[0];	// объявление переменных

for (i=0; i<10; i++)
{
  if (ans > arr[i])
  {
    ans = arr[i];
    k = i;
  }
}
	// цикл поиска
	
	// проверка элемента
	
	// присвоение значений

Если в массиве несколько элементов имеют минимальное значение, то в k будет запоминаться индекс первого из них. Если проверять условие *ans >= arr[i]*, то будет запоминаться индекс последнего элемента. Для поиска максимального элемента нужно проверять условие *ans < arr[i]*.

8. Алгоритм инверсии

Важной задачей при составлении сложных алгоритмов является инвертирование вектора, например, если задан одномерный массив A $[1 \quad 2 \quad 3 \quad 9 \quad 7 \quad 1]$, то инверсный массив A^I будет выглядеть следующим образом $[1 \quad 7 \quad 9 \quad 3 \quad 2 \quad 1]^I$.

Алгоритм, реализующий инверсию, приведен в таблице 4.12.

Таблица 4.12. Способ инверсии массива

Код	Комментарии
int i, buf; int array[7] = {1,7,9,3,2,1,0};	// объявление // переменных

for (i=0; i<(7/2); i++)
{
  buf = array[i];
  array[i] = array[7-1-i];
  array[7-1-i] = buf;
}
	// инверсия

Вопросы для самопроверки

1. Является ли массив сложным типом данных?
2. Может ли массив включать в себя элементы сложного типа данных?
3. Можно ли заполнить массив данными «в одну строку кода» соблюдая при этом «правила хорошего тона»?
4. Как осуществляется доступ к элементам массива?

Практические задания

1. Необходимо написать программу на языке Си, которая вводит массив из 10 элементов, а затем выводит из него те элементы, которые являются четными.

2. Необходимо написать программу на языке Си, которая вводит массив из 25 элементов, а затем определяет по ним среднее значение и дисперсию по формулам:

$$m = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad S^2 = \frac{1}{N-1}\sum_{i=1}^{N}(m - x_i)^2$$

3. Необходимо написать программу на языке Си, которая вводит массив из 8 элементов, затем находит среднее значение элементов этого массива и преобразует исходный массив, вычитая из каждого элемента среднее значение.

4. Необходимо написать программу на языке Си, которая вводит массив из 10 элементов и вычисляет сумму его элементов, расположенных до первого отрицательного элемента.

Практическая работа № 4. Обработка одномерных массивов

Цель работы – овладение практическими навыками работы с массивами, изучение особенностей их ввода и вывода.

Задание к работе

В соответствии с вариантом задания (таблица 4.13), в среде Visual Studio необходимо разработать программу, выполняющую:

- формирование массивов входных величин в соответствии с вариантом задания;
- обработку массивов;
- вывод в консоль исходных массивов и результатов обработки.

Нужно составить программу таким образом, чтобы она была применима для массивов любых размеров в пределах тех ограничений, которые указаны в задаче.

Таблица 4.13. Варианты заданий к практической работе № 4

№ вар.	Формулировка задания	Условия и ограничения
1	Найти первый положительный элемент массива $x[N]$, изменить у него знак и поставить в начало массива.	$N \le 50$; $-10 \le x_i \le 10$
2	Переписать отрицательные элементы массива $x[20]$ в массив y и подсчитать их количество.	$-1 \le x_i \le 1$
3	Каждый третий элемент массива $a[N]$ заменить квадратом индекса.	$N \le 25$; $0 \le a_i \le 1$
4	Найти последний положительный элемент массива $a[40]$, величина которого не превышает заданной величины b.	$-10 \le a_i \le 10$; $b = 5$
5	Из элементов массива $a[K]$, не превышающих заданную величину b, сформировать массив $y[]$.	$K \le 20$; $a_i \ge 0$; $b = 3$
6	Найти минимальный элемент массива $x[K]$, сменить у него знак и поставить в конец массива.	$K \le 30$; $-10 \le a_i \le 10$
7	Переписать подряд в массив $y[]$ положительные и в $z[]$ отрицательные элементы массива $x[N]$.	$N \le 40$
8	Найти последний отрицательный элемент массива $a[i]$ и заменить его индексом i.	$L \le 25$; $-5 \le a_i \le 5$
9	Сформировать массив $y[]$ из положительных элементов массива $x[N]$, величины которых находятся в заданном диапазоне $[a, b]$.	$N \le 30$; $a > b$
10	Найти максимальный элемент массива $c[K]$ и его порядковый номер.	$K \le 20$; $-1 \le c_i \le 1$
11	Сформировать массив $y[]$ из элементов массива $x[20]$, модели которых не превышают заданную величину b.	$-5 \le x_i \le 5$; $b = 1$
12	Расположить в массиве $r[]$ сначала положительные, а затем отрицательные элементы массива $z[N]$.	$N \le 30$;
13	Вычислить среднее арифметическое значение элемента массива $a[K]$.	$K \le 25$; $a_i \ge 0$
14	Увеличить длину массива $x[N]$, вставив заданное число a перед минимальным элементом массива.	$N \le 30$; $a = 5$
15	Определить сумму элементов массива $r[M]$, кратных трем.	$M \le 20$;

№ вар.	Формулировка задания	Условия и ограничения
16	Уменьшить длину массива $a[30]$, удалив из массива максимальный элемент.	-
17	Найти максимальный и минимальный элементы массива $d[K]$ и поменять их местами.	$K \le 15$
18	Найти минимальный положительный элемент массива $a[30]$ и заменить его индексом.	$-1 \le a_i \le 1$
19	Найти максимальный отрицательный элемент массива $x[N]$ и заменить его индексом	$N \le 30$; $-10 \le x_i \le 10$
20	Сформировать массив $b[]$ из элементов исходного массива $a[N]$, модули которых меньше заданной величины f.	$N \le 20$; $b = 5{,}5$

Порядок выполнения работы

1. Изучить:

- способы создания массивов на языке Си/Си++;
- способы ввода и вывода массивов.

2. Разработать алгоритм решения задачи в соответствии с заданием.

3. Составить программу решения задачи. Программа должна включать в себя три составляющие: блок ввода (генерации) данных, блок обработки данных, блок вывода результатов обработки.

4. Набрать программу в редакторе системы программирования Microsoft Visual Studio C++.

5. Протестировать разработанную программу. При наличии ошибок внести изменения в программу.

6. Оформить отчет к работе. Отчет должен содержать:

- цель работы;
- задание к работе (с указанием варианта);
- описание алгоритма;
- исходный код программы на языке Си/Си++ (с комментариями);

- результаты тестирования (в том числе скриншот работы программы);

- выводы.

Контрольные вопросы и задания

1. Как создать массив на языке Си/Си++?
2. Как осуществляется доступ к элементам массива?
3. Укажите особенности консольного ввода и вывода массивов.
4. Как вывести массив в столбец и как вывести его в строку?

4.3. Многомерные массивы

Рассмотренные выше примеры относятся к одномерным массивам (векторам). Другим видом массива является матрица – двухмерный массив.

Двухмерный массив объявляется следующим образом (таблица 4.14).

Таблица 4.14. Пример объявления одномерного и двухмерного массивов

Код	Комментарии
int array[5];	// одномерный массив (вектор)
int matrix[5][5];	// двухмерный массив (матрица)

Приведенный в примере массив matrix занимает в ОЗУ 100 байт и включает в себя в общей сложности 25 элементов, от элемента [0][0] и до элемента [4][4].

Использование матриц получило широкое распространение в математике и активно используется в программировании.

Рассмотрим пример: необходимо создать матрицу размером 3x3 состоящую из случайных чисел, а затем вывести ее на экран. Программа решения этой задачи приведена в таблице 4.15.

Таблица 4.15. Пример генерации матрицы и вывода ее в консоль

Код	Комментарии
#include "stdafx.h" #include <stdio.h> #include <stdlib.h>	// подключение библиотек
int main(void) {	// основная процедура
int i,j;	// простые переменные
int matrix[3][3];	// двухмерный массив

for (i=0; i<3; i++)
for (j=0; j<3; j++)
matrix[i][j] = rand() % 100;
	// для работы с матрицами
	// часто используют
	// вложенные циклы

for (i=0; i<3; i++)
{
for (j=0; j<3; j++)
{
printf("%02d ", matrix[i][j]);
}
printf("\n");
}
	// вывод данных

fflush(stdin); getchar();	// пауза
return 0; }	// завершение программы

В предлагаемой программе используется новая функция – **rand()**, которая входит в состав библиотеки **stdlib.h** и в составе инструкции **rand() % 100** служит для генерации целых случайных чисел в диапазоне от «0» до «99».

Следует обратить особое внимание и запомнить: первый индекс матрицы matrix[i][j], обычно обозначается i или row – это строка (ряд) матрицы, а второй символ матрицы matrix[i][j], обычно обозначается j или column – это столбец (колонка) матрицы.

Помимо одномерных и двухмерных массивов, по аналогии можно создавать любые *N*-мерные массивы, однако такие задачи на практике встречаются редко. Пример объявления трехмерного и четырехмерного массива представлен в таблице 4.16.

Таблица 4.16. Пример объявления трехмерного и четырехмерного массива

Код	Комментарии
int array_3[5][5][5];	// трехмерный массив
int array_4[5][5][3][3];	// четырехмерный массив

В первом примере массив array_3 занимает в ОЗУ 500 байт, а во втором примере array_4 занимает 900 байт.

Как можно видеть, для *N*-мерных массивов требуется достаточно много памяти даже при относительно небольших размерностях.

4.4. Алгоритмы типовых действий с двумерными массивами (матрицами)

1. Ввод матриц с клавиатуры

Пример реализации алгоритма приведен в таблице 4.17.

Таблица 4.17. Ввод матрицы с клавиатуры

Код	Комментарии
int i, j, matr[5][5];	// объявление переменных

for (i=0; i<5; i++)
{
for (j=0; j<5; j++)
{
scanf("%d", &matr[i][j]);
}
}
	// строки
	
	// столбцы
	
	// ввод данных

При работе с массивами часто возникает задача присвоить всем элементам массива одно и то же значение, чаще всего ноль или единицу.

В таблице 4.18 представлены алгоритмы обнуления массивов на примере одномерного массива и матрицы.

Таблица 4.18. Способ обнуления элементов массива

Код	Комментарии
int value = 0; int i,j; int array[10], matrix[5][5];	// объявление переменных
for (i=0; i<10; i++) array[i] = value;	// обнуление массива

for (i=0; i<5; i++)
  for (j=0; j<5; j++)
    matrix[i][j] = value;
	// обнуление матрицы

Если изменить значение переменной value с 0 на 1, то массив «array» и матрица «matrix» будут заполнены единицами.

Важной операцией при работе с матрицами является создание тензорной матрицы и ее частного случая – единичной матрицы, т.е. такой матрицы, все элементы которой – нулевые, за исключением главной диагонали, элементы которой единичные:

$$one = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad tensor = \begin{bmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & z \end{bmatrix}.$$

В таблице 4.19 предлагается алгоритм создания единичной и классической тензорной матрицы.

Таблица 4.19. Способы генерации единичной и тензорной матрицы

Код	Комментарии

int i,j;
int array[5] = {5,7,3,2,8};
int tensor[5][5];
int one[5][5];
	// простые переменные
	// элементы диагонали
	// тензорная матрица
	// единичная матрица

for (i=0; i<5; i++)
{
  for (j=0; j<5; j++)
    one[i][j] = 0;
  one[i][i]   = 1;
}
	// создание единичной матрицы
	
	// обнуление элементов
	// формирование диагонали

for (i=0; i<5; i++)
{
  for (j=0; j<5; j++)
    tensor[i][j] = 0;
  tensor[i][i]   = array[i];
}
	// создание тензорной матрицы
	
	// обнуление элементов
	// формирование диагонали

2. *Вывод матрицы на экран.*

Пример реализации алгоритма приведен в таблице 4.20.

Таблица 4.20. Вывод матрицы на экран

Код	Комментарии
`int i, j, matr[5][5];`	// объявление переменных

`for (i=0; i<5; i++)`
`{`
` for (j=0; j<5; j++)`
` {`
`  printf("%02d ", matr[i][j]);`
` }`
` printf ("\n");`
`}`
	// строки
	
	// столбцы
	
	// вывод данных

3. *Суммирование элементов матрицы*

Пример реализации алгоритма приведен в таблице 4.21.

Таблица 4.21. Суммирование элементов матрицы

Код	Комментарии
`int i, j, matr[5][5];` `int s = 0;`	// объявление переменных

`for (i=0; i<5; i++)`
`{`
` for (j=0; j<5; j++)`
` {`
`  s += matr[i][j];`
` }`
`}`
	// строки
	
	// столбцы
	
	// суммирование

4. *Суммирование диагональных элементов матрицы* (вычисление следа матрицы)

Пример реализации алгоритма приведен в таблице 4.22.

Таблица 4.22. Суммирование диагональных элементов матрицы

Код	Комментарии
int i, matr[5][5]; int s = 0;	// объявление переменных

for (i=0; i<5; i++)
{
  s += matr[i][i];
}
	// вложенный цикл не требуется
	
	// суммирование

5. *Суммирование двух матриц по элементам* (обе матрицы должны полностью совпадать как по размерности, так и по типам их элементов).

Пример реализации алгоритма приведен в таблице 4.23.

Таблица 4.23. Суммирование двух матриц по элементам

Код	Комментарии
int i, j, a[5][5], b[5][5], c[5][5];	// объявление переменных

for (i=0; i<5; i++)
{
  for (j=0; j<5; j++)
  {
    c[i][j] = a[i][j] + b[i][j];
  }
}
	// строки
	
	// столбцы
	
	// суммирование

6. *Суммирование элементов строк матриц*

Необходимо вычислить сумму элементов каждой строки матрицы c размерами $N \times M$. Результат получим в виде вектора d.

Пример реализации алгоритма приведен в таблице 4.24.

Таблица 4.24. Суммирование элементов строк матриц

Код	Комментарии
int i, j, c[5][5]; int s, d[5];	// объявление переменных

for (i=0; i<5; i++)
{
  s = 0;
  for (j=0; j<5; j++)
  {
    s += matr[i][j];
  }
  d[i] = s;
}
	// строки
	
	// обнуление суммы
	// столбцы
	
	// суммирование
	
	// запись результата в массив
	// (вектор)

7. *Транспонирование матрицы*

Для квадратной матрицы размерами $N \times N$ для этого необходимо поменять местами каждый элемент верхнего треугольника с соответствующим элементом нижнего (диагональные элементы переставлять не нужно). При этом для каждой строки нужно выполнять перестановку элементов, расположенных правее главной диагонали, с элементами соответствующего столбца, расположенного ниже главной диагонали, т.е. взаимно перемещать элементы $C_{i,j}$ и $C_{j,i}$,i=1, ...,N-1; j = i+1, ... , N.

Пример реализации алгоритма приведен в таблице 4.25.

Таблица 4.25. Транспонирование матриц

Код	Комментарии
int i, j, buf, c[5][5];	// объявление переменных

for (i=0; i<(5-1); i++)
{
  for (j=i+1; j<5; j++)
  {
    buf = c[i][j];
    c[i][j] = c[j][i];
    c[j][i] = buf;
  }
}
	// строки
	
	// столбцы
	
	// замена элементов между
	// собой через буфер

8. *Умножение матрицы на вектор:*

Для вычисления произведения C матрицы A размерами $N \times M$ на вектор B размером M необходимо вычислить $C_i = \sum_{j=1}^{M} a_{ij} b_j$, $i=1, \ldots N$. Пример реализации алгоритма приведен в таблице 4.26.

Таблица 4.26. Умножение матрицы на вектор

Код	Комментарии
`int i, j, matr[5][5];` `int s, b[5], c[5];`	// объявление переменных

`for (i=0; i<5; i++)`
`{`
`  s = 0;`
`  for (j=0; j<5; j++)`
`  {`
`    s += matr[i][j] * b[j];`
`  }`
`  c[i] = s;`
`}`
	// строки
	
	// обнуление суммы
	// столбцы
	
	// накопление суммы
	
	// запись суммы в массив

9. *Умножение матрицы на матрицу:*

Для умножения матрицы A размерами $N \times L$ на матрицу B размерами $L \times M$ необходимо вычислить $C_{ij} = \sum_{k=1}^{L} a_{ik} b_{kj}$, $i = 1 \ldots N$, $j = 1, \ldots M$.

Пример реализации алгоритма приведен в таблице 4.27.

Таблица 4.27. Умножение матрицы на матрицу

Код	Комментарии
int i, j, k, a[5][5], b[5][5]; int s, c[5][5];	// объявление переменных

for (i=0; i<5; i++)
{
 for (j=0; j<5; j++)
 {
  s = 0;
  for (k=0; k<5; k++)
  {
   s += a[i][k] * b[k][j];
  }
  c[i][j] = s;
 }
}
	// строки
	
	// столбцы
	
	// запись результата в новую матрицу

10. *Удаление строки из матрицы*

Требуется удалить строку с заданным номером *K*. Решение этой задачи аналогично решению задачи удаления элемента из одномерного массива. Все строки, начиная с *(K* + 1)-й, нужно переместить вверх. Число строк уменьшается на 1.

Удаление столбца осуществляется аналогично.

Пример реализации алгоритма приведен в таблице 4.28.

Таблица 4.28. Удаление элементов из матрицы

Код	Комментарии
int i,j, index; int matrix_1[3][3] = {…}; int matrix_2[3][3] = {…};	// объявление // переменных

index = 1;
for (i=index; i<(3-1); i++)
 for (j=0; j<3; j++)
  matrix_2[i][j] = matrix_2[i+1][j];
	// удалена будет
	// строка с индексом // 1, соответственно [4,2,1]

index = 0;
for (i=index; i<(3-1); i++)
 for (j=0; j<3; j++)
  matrix_1[j][i] = matrix_1[j][i+1];
	// удален будет
	// столбец с индексом // 0, соответственно [1;4;3]

Следует особо отметить, что удаление элемента в одномерном массиве, строки или столбца в матрице не приводит к изменению ко-

личества элементов массивов. Так однажды объявленная матрица размером 3x3 сохраняет свой размер на протяжении всего периода ее существования. Например, если из матрицы размером 3x3 $\begin{bmatrix} 1 & 3 & 7 \\ 4 & 2 & 1 \\ 3 & 8 & 9 \end{bmatrix}$ удалить вторую строку, то в результате получим новую матрицу $\begin{bmatrix} 1 & 3 & 7 \\ 3 & 8 & 9 \\ 3 & 8 & 9 \end{bmatrix}$.

Аналогично, если из матрицы размером 3x3 $\begin{bmatrix} 1 & 3 & 7 \\ 4 & 2 & 1 \\ 3 & 8 & 9 \end{bmatrix}$ удалить первый столбец, то в результате получим новую матрицу $\begin{bmatrix} 3 & 7 & 7 \\ 2 & 1 & 1 \\ 8 & 9 & 9 \end{bmatrix}$.

11. *Включение строки в матрицу*

Включаемая строка задана как вектор *C*. Пример реализации алгоритма приведен в таблице 4.29.

Таблица 4.29. Включение строки в матрицу

Код	Комментарии
int i, j, b[6][5], c[5], k = 1;	// объявление переменных

for (i=(5-1); i>=k; i--)
{
  for (j=0; j<5; j++)
  {
    b[i+1][j] = b[i][j];
  }
}
for (j=0; j<5; j++)
{
  b[k][j] = c[j];
}
	// строки
	
	// столбцы
	
	// сдвиг строк матрицы
	
	// добавление новой строки
	// в матрицу

12. *Поиск минимального (максимального) элемента в матрице*

Требуется просматривать все элементы матрицы, что требует организации двойного цикла, и запоминать два индекса: номер строки *K* и номер столбца *L*.

Пример реализации алгоритма приведен в таблице 4.30.

Таблица 4.30. Поиск минимального элемента в матрице

Код	Комментарии
int i, j, c[5][5]; int k, l, ans = 1e6;	// объявление переменных

for (i=0; i<5; i++)
{
  for (j=0; j<5; j++)
  {
    if (ans > c[i][j])
    {
      ans = c[i][j];
      k = i;
      l = j;
    }
  }
}
	// строки
	
	// столбцы
	
	// поиск минимума
	
	// присвоение значений

13. *Преобразование матрицы*

Преобразование матрицы, как правило, производится относительно какой-либо оси, которой могут быть главная диагональ, побочная диагональ, любая строка или столбец.

Рассмотрим эти понятия на примере матрицы A размерами $N \times N$, обозначив номер строки переменной I, а номер столбца – переменной J.

1. Элементы главной диагонали имеют одинаковые индексы, т.е. $A(1,1)$ (например $A(1,1)$, $A(2, 2)$,... $A(N, N)$).

2. У всех элементов, расположенных выше (правее) главной диагонали, $J>I$.

3. У всех элементов, расположенных ниже главной диагонали, $J<I$.

4. Элементами, симметричными относительно главной диагонали, являются пары $A(I, J)$ и $A(J, I)$.

5. У элементов, образующих побочную диагональ, есть зависимость индексов: $I+J=N+1$, или $J=N+1-I$, т.е. $A(I, N+1-I)$.

6. У всех элементов, расположенных выше побочной диагонали, $I<N+1-J$.

7. У всех элементов, расположенных ниже побочной диагонали, $I>N+1-J$.

8. Элементами, симметричными относительно побочной диагонали, являются пары $A(I, J)$ и $A(N+1-J, N+1-I)$.

9. Элементами, симметричными относительно средней горизонтальной оси, являются пары $A(I, J)$ и $A(I, N+1-J)$.

Вопросы для самопроверки

1. Как осуществляется объявление и доступ к матрице?
2. Как осуществляется обнуление матрицы?
3. Что такое тензорная матрица и как ее получить?
4. Как осуществляется удаление строки матрицы?
5. Как добавить новую строку в матрицу?

Практические задания

1. Необходимо написать программу на языке Си, которая генерирует две матрицы размером 3х3, производит их сложение и вывод результата на экран.
2. Необходимо написать программу на языке Си, которая генерирует матрицу размером 4х4, производит умножение этой матрицы на число и вывод результата на экран.
3. Задана матрица размерами $n \times m$. Просуммировать положительные элементы каждой строки. Результат получить в виде вектора размером n.
4. Для матрицы размерами $n \times m$ найти максимальный элемент каждой строки. Результат получить в виде вектора размером n.
5. В квадратной матрице размерами 5×5 исключить строку и столбец, на пересечении которых расположен максимальный элемент главной диагонали.

Практическая работа № 5. Обработка матриц

Цель работы – овладение навыками программирования алгоритмов, использующих операции с матрицами.

Задание к работе

В соответствии с вариантом задания (таблица 4.31), в среде Visual Studio необходимо разработать программу, выполняющую:

- формирование матриц входных величин;
- обработку матриц;
- вывод в консоль исходной матрицы и результатов обработки.

Программу нужно составить таким образом, чтобы она была применима для матриц любых размеров в пределах заданных ограничений: $N < 10$, $M < 10$.

Исходные данные и результаты вычисления необходимо вывести в консоль в общепринятом виде (матрицы необходимо выводить в том формате, который принят в математике).

Таблица 4.31. Варианты заданий к практической работе № 5

№ вар.	**Формулировка задания**
1	Смените знаки у элементов матрицы $a[N, M]$, лежащих выше и ниже главной диагонали.
2	Сменить знаки элементов строки и столбцы матрицы $a[N, M]$, на пересечении которых находится минимальный элемент.
3	Найти максимальный и минимальный элементы матрицы $c[N, M]$ и поменять их местами.
4	Найти строки с наибольшей и наименьшей суммой элементов матрицы $t[N, M]$. Вывести на печать найденные строки и суммы их элементов.
5	Сформировать одномерный массив $y[]$, элементы которого равны максимальным модулям элементов строки матрицы $x[N, M]$.
6	Вычислить сумму и число элементов матрицы $b[N, M]$, находящихся под главной диагональю.
7	Уменьшить размер матрицы $c[N, M]$, удалив в каждой строке минимальные элементы.
8	Записать на место отрицательных элементов матрицы $d[N, M]$ нули, а вместо положительных – единицы. Вывести на печать нижнюю треугольную матрицу с полученными элементами.
9	Увеличить размер матрицы $a[N, M]$, добавив к ней строку, элементы которой равны суммам элементов в соответствующих столбцах.
10	Сменить знаки у элементов матрицы $b[N, M]$, лежащих выше главной диагонали и имеющих четную сумму индексов.
11	Найти минимальный элемент матрицы $d[N, M]$ и поставить его в начало того столбца, в котором он находится.
12	Найти максимальный по модулю элемент матрицы $f[N, M]$ и поставить его в начало той строки, в которой он находится.
13	Для целочисленной матрицы $a[N, M]$ найти для каждой строки число элементов, кратных пяти, и наибольший из полученных результатов.
14	В каждой строке матрицы $a[N, M]$ нулевые элементы переставить в конец строки.
15	Сформировать одномерный массив $y[]$, элементы которого представляют собой среднее арифметическое столбцов матрицы $x[N, M]$.

№ вар.	**Формулировка задания**
16	Найти в каждой строке матрицы *a*[*N*, *M*] минимальный элемент и поменять его местами с элементом главной диагонали.
17	Вычислить сумму и число элементов матрицы *b*[*N*, *M*], находящихся под главной диагональю и на ней.
18	Сформировать одномерный массив *x*[] из строк матрицы *y*[*N*, *M*] с минимальной суммой элементов.
19	Уменьшить размер матрицы *a*[*N*, *M*], удалив из нее строки с нулевыми элементами выше главной диагонали.
20	Увеличить размер матрицы *b*[*N*, *M*], добавив столбец, элементы которого равны количеству положительных элементов в соответствующих строках.

Порядок выполнения работы

1. Изучить:

- способы консольного ввода и вывода матриц;
- приемы программирования структур с вложенными циклами.

2. Разработать алгоритм решения задачи в соответствии с заданием.

3. Составить программу решения задачи. Программа должна включать в себя три составляющие: блок ввода (генерации) данных, блок обработки данных, блок вывода результатов обработки.

4. Набрать программу в редакторе системы программирования Microsoft Visual Studio C++.

5. Протестировать разработанную программу. При наличии ошибок внести изменения в программу.

6. Оформить отчет к работе. Отчет должен содержать:

- цель работы;
- задание к работе (с указанием варианта);
- описание алгоритма;
- исходный код программы на языке Си/Си++ (с комментариями);
- результаты тестирования (в том числе скриншот работы программы);
- выводы.

Контрольные вопросы и задания

1. Как организовать ввод матрицы по строкам?
2. Как сгенерировать матрицу, содержащую случайные числа?
3. Как организовать вывод матрицы по столбцам?
4. Укажите способы выхода из внутреннего цикла.
5. Укажите способ пропуска итерации внешнего цикла.

4.5. Строка как особый вид массива

С помощью компьютера часто приходится обрабатывать не только числа, но и строки символов, например имена, фамилии, названия и т.д.

Строка – это тип данных, значениями которого является произвольная последовательность символов алфавита.

Так как язык Си/Си++ по своему происхождению является языком системного программирования, то строковый тип данных в Си/Си++ как таковой отсутствует, а в качестве строк в Си/Си++ используются обычные одномерные массивы типа **char**.

Строка обладает всеми теми же свойствами, что и одномерный массив. Единственным отличием такого массива, будет являться последний элемент, в котором хранится специальный символ, обозначающий конец строки. Это символ с кодом нуль, его обозначают следующим символом – '\0'. При объявлении строки нуль-символ добавляется к ней автоматически.

Объявление строки в Си/Си++ имеет тот же синтаксис, что и объявление одномерного символьного массива:

Char имя[длина].

Длина строки указывается с учетом одного символа на хранение завершающего нуля, поэтому максимальное количество значащих символов в строке на единицу меньше ее длины. Например, строка может содержать максимум четырнадцать символов, если объявлена следующим образом: char str[15].

После того, как мы объявили массив символов, в нем будет записан мусор. Как и любая переменная, строка может быть инициализирована (т.е. ей присвоено некоторое значение) непосредственно при её объявлении:

char имя[длина] = "строковый литерал ".

Пример объявления строки приведен в таблице 4.32.

Таблица 4.32. Пример объявления строки

Код	Комментарии
int array[5] = {0, 1, 3, 9, 0};	// одномерный массив
char str[6] = "hello";	// строка (тоже массив)

Как видно из примера таблицы 4.32, первой особенностью строкового типа – является ее объявление: при создании строковой переменной можно указать последовательность символов в точно таком же виде, как в строке на листе бумаги.

Обратите внимание: длина строки может быть больше, чем длина текста хранящегося в строке.

Следует отметить, что строка обрамляется в двойные кавычки, тогда как одинарный символ – в одинарные кавычки.

Массивы строк в Си/Си++

Объявление массивов строк в языке Си/Си++ также возможно. Для этого используются двумерные массивы символов, что имеет следующий синтаксис:

char имя[количество строк][длина];

Первым индексом матрицы указывается количество строк в массиве, а вторым – максимальная (с учетом завершающего нуля) длина каждой строки. Например, объявление массива из пяти строк максимальной длиной 30 значащих символов будет иметь вид:

```
char strs[5][31];
```

При объявлении массивов строк можно производить инициализацию. При этом число строковых литералов должно быть меньше или равно количеству строк в массиве. Если число строковых литералов меньше размера массива, то все остальные элементы инициализируются пустыми строками. Длина каждого строкового литерала должна быть строго меньше значения длины строки (для записи завершающего нуля).

Например:

```
char months[12][10] = {
  "Январь", "Февраль", "Март", "Апрель", "Май",
 "Июнь", "Июль", "Август", "Сентябрь", "Октябрь",
 "Ноябрь", "Декабрь"
};
```

Так как строки на языке Си/Си++ являются массивами символов, то к любому символу строки можно обратиться по его индексу. Для этого используется синтаксис обращения к элементу массива, поэтому первый символ в строке имеет индекс ноль. Например, в следующем фрагменте программы в строке str осуществляется замена всех символов 'a' на символы 'A' и наоборот (см. таблицу 4.33).

Таблица 4.33. Пример работы к элементам строки по индексам

Код	Комментарии

for (int i = 0; str[i] != 0; i++)
{
    if (str[i] == 'a') str[i] = 'A';
    else if (str[i] == 'A') str[i] = 'a';
}
	// перебор элементов строки
	
	// замена строчных букв
	// заглавными и наоборот

Для ввода и вывода строковой информации можно использовать функции форматированного ввода и вывода (**scanf**() и **printf**()). Для вывода строки посредством функции **printf()**, используется специальная конструкция **%s**. Следует обратить особое внимание, что функции **scanf()**, при вводе строки не требуется знак & (амперсант).

Например, ввод и последующий вывод строковой переменной будет иметь вид (таблица 4.34).

Таблица 4.34. Пример ввода и вывода строки

Код	Комментарии
char str[31] = "";	// объявление переменных

printf("Enter string: ");
scanf("%30s",str);
printf("You enter: %s",str);
	
	// ввод строки
	// вывод строки в консоль

Недостатком функции **scanf**() при вводе строковых данных является то, что символами разделителями данной функции являются перевод строки, табуляция, пробел. Поэтому, используя данную функцию невозможно ввести строку, содержащую несколько слов, разделенных пробелами или табуляциями.

Для ввода и вывода строк так же можно использовать специализированные функции **fgets**() и **puts**(), которые находятся в библиотеке **stdio.h.**

Функция **fgets**() предназначена для ввода строк и имеет следующий заголовок:

```
char * fgets(char * buffer, int size, FILE * stream);
```

где buffer – строка для записи результата, size – максимальное количество байт, которое запишет функция fgets(), stream – файловый объект для чтения данных, для чтения с клавиатуры нужно указать stdin. Эта функция читает символы со стандартного ввода, пока не считает n-1 символ или символ конца строки, потом запишет считанные символы в строку и добавит нулевой символ. При этом функция fgets() записывает символ конца строки в данную строку, что нужно учитывать.

Функция **puts**() предназначена для вывода строк и имеет следующий заголовок:

int puts(const char *string);

Простейшая программа ввода и вывода строки с использованием функций **fgets**() и **puts**() представлена в таблице 4.35.

Таблица 4.35. Пример использование функций fgets и puts

Код	Комментарии
char str[50] = "";	// объявление переменных

printf("Enter string: ");
fgets(str, 50, stdin);
printf("You enter: ");
puts(str);
	
	// ввод строки
	
	// вывод строки в консоль

Для работы со строками доступны все те же операции, что и для массива, однако имеется и ряд дополнительных возможностей. Для того чтобы воспользоваться ими необходимо в начале программы подключить библиотеку **string.h**. В таблице 4.36 приводится перечень наиболее важных функций этой библиотеки.

Таблица 4.36. Основные функции библиотеки string.h

Функция	Описание
char *strcat(char *s1, char *s2)	присоединяет s2 к s1, возвращает s1
char *strncat(char *s1, char *s2, int n)	присоединяет не более n символов s2 к s1, завершает строку символом '\0', возвращает s1
char *strcpy(char *s1, char *s2)	копирует строку s2 в строку s1, включая '\0', возвращает s1
char *strncpy(char *s1, char *s2, int n)	копирует не более n символов строки s2 в строку s1, возвращает s1;
int strcmp(char *s1, char *s2)	сравнивает s1 и s2, возвращает значение 0, если строки эквивалентны

Функция	Описание
int strncmp(char *s1, char *s2, int n)	сравнивает не более n символов строк s1 и s2, возвращает значение 0, если начальные n символов строк эквивалентны
int strlen(char *s)	возвращает количество символов в строке s
char *strset(char *s, char c)	заполняет строку s символами, код которых равен значению c, возвращает указатель на строку s
char *strnset(char *s, char c, int n)	заменяет первые n символов строки s символами, код которых равен c, возвращает указатель на строку s

Вопросы для самопроверки

1. Как осуществляется объявление строки в программе?
2. Как инициируется строка в программе?
3. Какие функции используются для ввода строки?
4. Какие функции используются для вывода строки?
5. Какие функции для работы со строками Вы знаете?

Практические задания

1. Необходимо написать программу на языке Си, которая в заданной строке удалит второй и четвертый по счету символы.
2. Необходимо написать программу на языке Си, которая удаляет первое слово заданной строки. Разделителем слов считается пробел.
3. Необходимо написать программу на языке Си, которая для заданной строки определит входит ли в нее хотя бы одна латинская буква.
4. Необходимо написать программу на языке Си, которая в заданной строке заменяет каждый пробел двумя пробелами.
5. Необходимо написать программу на языке Си, которая меняет в строке круглые скобки на квадратные.

Практическая работа № 6. Обработка символьных и строковых данных

Цель работы – овладение практическими навыками работы с символьными и строковыми типами данных.

Задание к работе

В соответствии с вариантом задания (таблица 4.37), в среде Visual Studio необходимо разработать программу, выполняющую:

- формирование строки типа char[] (см. вариант задания);
- обработку строки;
- вывод в консоль исходной строки и результатов ее обработки.

Таблица 4.37. Варианты заданий к практической работе № 6

№ вар.	Формулировка задания
1	Удалить из текста пробелы и подсчитать длину сформированного текста.
2	Проверить, имеется ли в заданном тексте баланс открывающих и закрывающих скобок.
3	Подсчитать, сколько имеется в тексте пар одинаковых букв.
4	Поменять в тексте квадратные скобки на круглые. Подсчитать количество замен.
5	Удалить из текста символ “ (двойные кавычки) и подсчитать количество оставшихся символов.
6	Из текста выбрать цифры и записать их в массив $x[10]$.
7	Найти в тексте симметричные слова. Например, ABBA.
8	Отредактировать текст, заменяя многоточия точкой. Найти длину измененного текста.
9	Отредактировать текст, удаляя из него лишние пробелы. Заключить весь текст в кавычки.
10	В заданном тексте подсчитать количество гласных букв латинского алфавита.
11	Из заданного текста выбрать и напечатать только те символы, которые встречаются в нем один раз.
12	Вставить в заданный текст недостающие закрывающиеся скобки.
13	В заданном выражении поменять местами знаки умножения и деления.
14	Подсчитать в тексте количество знаков препинания.
15	Подсчитать в выражении количество арифметических операций.
16	Заключить в скобки цифры в заданном тексте.
17	Подсчитать в тесте количество запятых. Поставить в конец текста многоточие.
18	Создать массив букв. Подсчитать в заданном тексте количество букв, входящих в созданный массив.
19	Создать массив цифр. Подсчитать в арифметическом выражении количество цифр, входящих в созданный массив.
20	Найти сумму всех цифр, используемых в тексте.

Порядок выполнения работы

1. Изучить:

- способы описания символьных и строковых данных;
- стандартные процедуры и функции, определенные над символьным и строковыми типами значений.

2. Разработать алгоритм решения задачи в соответствии с заданием.

3. Составить программу решения задачи. Программа должна включать в себя три составляющие: блок ввода данных, блок обработки данных, блок вывода результатов обработки.

4. Подготовить тестовую строку для проверки правильности функционирования программ.

5. Набрать программу в редакторе системы программирования Microsoft Visual Studio C++.

6. Протестировать разработанную программу. При наличии ошибок внести изменения в программу.

7. Оформить отчет к работе. Отчет должен содержать:

- цель работы;
- задание к работе (с указанием варианта);
- описание алгоритма;
- исходный код программы на языке Си/Си++ (с комментариями);
- тестовое задание;
- результаты тестирования (в том числе скриншот работы программы);
- выводы.

Контрольные вопросы и задания

1. Как создаются символьные переменные?

2. Как создаются строки?

3. Чем строки отличаются от массивов?

4. Перечислите основные функции для работы со строковыми переменными из стандартной библиотеки string.h.

5. Назовите основные преимущества стандартных функций библиотеки string.h над пользовательскими функциями аналогичного назначения.

4.6. Структуры данных

Определение структуры данных

Выше рассматривался такой сложный тип данных, как массив. Массив, как известно, состоит из набора элементов одного типа. Однако зачастую требуется использовать в программе такой сложный тип данных, который включает в себя элементы одновременно нескольких типов, например, для представления студента потребуется указать: номер его зачетной книжки, фамилию, имя, отчество, группу, в которой он учится, дату рождения и т.п. В языке Си/Си++ для работы с такими объектами используется тип данных, который называется структура.

Структура – композитный тип данных, включающий в себя набор элементов разных типов.

Структуры позволяют трактовать группу связанных между собой объектов не как множество отдельных элементов, а как единое целое.

Объявление структур

В отличие от массивов, а также всех простых типов данных структура требует предварительного описания (объявления), т.к. заранее неизвестно, какие элементы программист хочет поместить в структуру.

Общий вид объявления типа данных структура выглядит следующим образом:

```
struct имя тип_структуры
{
    тип ИмяЭлемента1;
    тип ИмяЭлемента2;
   . . .
    тип ИмяЭлементаn;
};
```

Объявление структуры производится вне функций и выполняется посредством ключевого слова **struct**.

После закрывающей фигурной скобки } в объявлении структуры обязательно ставится точка с запятой.

В таблице 4.38 приведено описание структурированного типа данных с именем user_struct, состоящего из пяти элементов (полей): поля с именем symbol типа char, двух полей с именами i и j типа int, поля с именем value типа float и поля is типа bool.

Таблица 4.38. Пример объявление структурированного типа данных

Код	Комментарии
struct user_struct {	// объявление структуры
char symbol; int i,j; float value; bool is;	// элементы структуры
};	// в конце ставится ";"!

Особенности объявления структур:

- Присвоение значений при объявлении элементов структуры **запрещается**.
- Объявление структуры не создает никаких экземпляров! Создается только новый, структурированный тип данных (в примере – это «user_struct»).

После того, как структура объявлена, можно создавать любое количество экземпляров структуры, точно также как создаются переменные за исключением того, что перед типом структуры необходимо указать ключевое слово **struct,** которое указывает компилятору, что этот тип является структурированным (таблица 4.39). Такая особенность объявления экземпляров структуры характерна только для Си, но не для Си++ (в Си++ слово struct не используется).

Таблица 4.39. Объявление экземпляра структуры

Код	Комментарии
struct user_struct st_1;	// создана переменная с именем st_1
struct user_struct st_2 ={'A', 1, 2, 5.5, true};	// создана переменная с именем st_2

Особенности создания экземпляров структур:

- Можно создавать не только отдельные экземпляры структур, но и массивы структур, по аналогии с массивами любых других типов (таблица 4.40).

Таблица 4.40. Пример создания массива экземпляров структур

Код	Комментарии
struct user_struct st_arr[10];	// st_arr – это массив из // 10 экземпляров user_struct

- Элементы структуры могут быть как простого типа, так и сложного, например массивом или структурой или даже массивом структур (таблица 4.41).

Таблица 4.41. Пример структуры включающей в себя массив структур

Код	Комментарии
struct struct_of_structs	// объявление структурированного типа
{	
int i;	// поле простого типа
int arr[10];	// поле типа «массив»
struct user_struct st;	// поле типа «структура»
struct user_struct st_arr[5];	// поле типа «массив структур»
};	

Обращение к элементам структуры

Структура имеет одно очень важное отличие от массива – ее элементы могут занимать в памяти компьютера разный объем.

В качестве примера, рассмотрим наш структурированный тип «user_struct» и его экземпляр st_2 (таблица 4.39). В таблице 4.42 приведена область памяти, в которой хранится значение экземпляра структуры с именем st_2.

Как видно из таблицы 4.42, переменная st_2 занимает в памяти 14 байт, причем одни ее элементы занимают 1 байт памяти, а другие 4 байта. Из-за такой особенности обращение к элементам структуры по индексу не представляется возможным.

Таблица 4.42. Хранение в памяти компьютера экземпляра структуры st_2

Тип элемента	char	int				int				float				bool
Значение элемента	'A'	1				2				5.5				true
Сдвиг адреса памяти, байт	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Доступ к элементам структуры осуществляется с помощью составного имени, которое состоит из имени переменной и имени элемента структуры, разделенные точкой, например st_1.symbol, st_1.i, st_1. Value.

Элементы экземпляра структуры работают так же, как и простые переменные, поэтому с ними можно выполнять все операции, которые определены для типов данных, к которым они относятся.

Инициализация экземпляра структуры

Инициализировать экземпляр структуры, означает задать значения всем элементам структуры. Присвоить значения элементам структуры можно следующими способами:

1. в момент создания экземпляра структуры (вторая строка таблицы 4.39);
2. с помощью оператора присваивания;
3. с использованием функций ввода-вывода (например, **printf()** и **scanf()**).

В таблице 4.43 приводится пример программы использующей структуры.

Таблица 4.43. Пример объявления и доступа к полям экземпляра структуры

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение библиотек
struct user_struct {	// объявление структуры // вне функций
char symbol; int i,j; float value;	

Код	Комментарии
`  bool is;` `};`	
`int main(void)` `{`	// основная процедура
`  struct user_struct st;`	// объявление переменной

`  st.is = true;`
`  st.i  = 0;`
`  st.j  = 0;`
`  st.value = 9.9;`
	// присвоение значений
	// элементам структуры

`  if (st.is == true)`
`  {`
`    printf("%.1f\n", st.value);`
`    printf("%d %d\n", st.i, st.j);`
`  }`
	// чтение из элемента
	// структуры

`  fflush(stdin);` `  getchar();`	// пауза
`  return 0;` `}`	// завершение программы

Структура, как объект

Язык Си не является объектно-ориентированным, но все же «структура» является частным примером «класса» – сложного типа данных характерного только для объектно-ориентированных языков.

Классы отличают следующие основные черты:

- возможность включения в класс не только простых и сложных типов данных, но и функций (например, sin() и cos());
- возможность скрытия значений, таким образом, обращение к отдельным элементам класса возможно лишь непосредственно из внутренних функций класса;
- наличие конструктора и деструктора – особых функций выполняющих предварительную подготовку экземпляра класса при его создании, а также при его уничтожении и ряд других особенностей.

Основополагающими понятиями объектно-ориентированного подхода является инкапсуляция, наследование и полиморфизм.

Инкапсуляция – это объединение в единое целое данных и алгоритмов обработки этих данных. В рамках объектно-ориентированного программирования данные называются полями

объекта, а алгоритмы – объектными методами. Инкапсуляция позволяет в наибольшей степени изолировать объект от внешнего окружения. Другим немаловажным следствием инкапсуляции является легкость обмена объектами, переноса их из одной программы в другую.

Наследование – это свойство объектов порождать своих потомков. Объект-потомок автоматически наследует от родителя все поля (переменные) и методы (функции), может дополнять объекты новыми полями и заменять (перекрывать) методы родителя или дополнять их.

Полиморфизм – это свойство родственных объектов (т.е. объектов, имеющих одного общего родителя) решать схожие по смыслу проблемы разными способами.

Наследование элементов структуры

Наследование является важным приемом не только при реализации объектно-ориентированного подхода, но и в структурном программировании, когда требуется создать несколько с виду одинаковых, но отличающихся всего несколькими элементами структур.

В качестве примера базовой (родительской) структуры, возьмем, созданный ранее структурированный тип user_struct. Предположим, что нам требуется создать два экземпляра структуры, один типа user_struct, а другой имеющий все те же поля, что и user_struct, но дополненный текстовым полем char str[50].

Согласно с классическим подходом нам придется создавать новый структурированный тип, а затем создавать его экземпляр. Однако благодаря наследованию можно значительно упростить эту процедуру.

В таблице 4.44 приводится пример описывающий оба эти подхода.

Таблица 4.44. Пример наследования полей родительской структуры

Код	Комментарии

struct user_struct
{
  char symbol;
  int i,j;
  float value;
  bool is;
};
	// объявление структуры
	// родителя

Код	Комментарии

struct classic
{
  char symbol, str[50];
  int i,j;
  float value;
  bool is;
};
	// объявление еще одной
	// структуры
	// (классический подход)

struct child : user_struct
{
  char str[50];
};
	// объявление структуры
	// наследника (ООП)

struct classic st_1; struct child st_2;	// обе структуры содержат // идентичные поля

Примеры структур

Если массивы широко применяются в программировании для решения различных математических задач, то структуры часто применяются для описания объектов реального мира. В качестве примера можно рассмотреть домашних животных, объединенных под родительской структурой pets (таблица 4.45). Структурами наследниками в таком случае будут являться, например хомяк (hamster), кошка (cat) и собака (dog).

Таблица 4.45. Пример структурированных типов «животные»

Код	Комментарии

struct pets
{
  char name[25]; // имя
  short paws;   // число лап
};
	// объявление
	// структуры
	// родителя

struct hamster : pets
{
  char color[10];
  int littleness;
  bool big_tooths;
};
	// хомяк

struct cat : pets
{
  char color[10];
  bool tail;
	// кошка

Код	Комментарии
bool mustache; bool wildness; };	

struct dog : pets
{
char color[10];
bool tail;
bool curls;
bool protector;
};
	// собака
	
	// кудрявая
	// защитник

struct cat tom; strcpy(tom.name, "Tom");	// создаем кота Тома // заполняем поля...

Вопросы для самопроверки

1. Что такое структура?
2. В чем состоит основное отличие структуры от массива?
3. Как осуществляется доступ к элементам структуры?
4. Чем структура отличается от класса?
5. Что такое наследование? Приведите пример.

Практические задания

1. Необходимо написать программу на языке Си, в которой задан массив структур, содержащий следующие поля: наименование_товара, количество, цена, дата_продажи. Определить количество товаров, которые проданы менее года назад и вывести сведения о них.

2. Необходимо написать программу на языке Си, в которой задан массив структур, содержащий следующие поля: ФИО, должность, зарплата. Вывести сведения о сотрудниках, у которых зарплата выше средней и возраст менее 30-и лет.

3. Необходимо написать программу на языке Си, в которой задан массив структур, содержащий следующие поля: автор книги, количество страниц, тираж. Вывести в консоль данные о книгах, в которых количество страниц больше 150.

4. Необходимо написать программу на языке Си, в которой задан массив структур, содержащий следующие поля: производитель ЦПУ, объем ОЗУ, видеокарта, цена. Вывести в консоль сведения о самом дешевом компьютере на базе ЦПУ AMD.

Глава 5. ПОДПРОГРАММЫ, ФУНКЦИИ, МОДУЛИ

5.1. Понятие подпрограммы, функции и процедуры

Часто в разных местах программы приходится выполнять по сути один и тот же алгоритм, но с различными данными. Для обеспечения большей компактности и повышения наглядности программы такой частичный алгоритм целесообразно выделить из основного текста программы и записать его только один раз, оформив его в виде самостоятельного программного объекта – *подпрограммы*.

Подпрограмма – поименованная (названная именем) часть компьютерной программы, содержащая описание определенного набора действий. Подпрограмма может быть многократно вызвана из разных частей программы.

Таким образом, подпрограммы представляют собой инструмент, с помощью которого любая программа может быть разбита на ряд в известной степени независимых друг от друга частей.

Подпрограмма должна быть описана до того, как она будет использована в программе или другой подпрограмме.

В программировании выделяют два вида подпрограмм: процедуры и функции.

Отличие функции от процедуры заключается в том, что результатом исполнения операторов, образующих тело функции, всегда является некоторое значение.

Функция – это подпрограмма специального вида, которая, кроме получения параметров, выполнения действий, и передачи результатов работы через параметры имеет еще одну особенность – она всегда возвращает результат.

Функция может *принимать параметры* и *возвращать значение.* Данные, передаваемые в функцию для обработки, называются *параметрами,* а результат вычислений функции ее *значением*.

Процедура – это независимый именованный блок, который после однократного описания можно многократно вызвать по имени. В Си/Си++ процедурой называют void-функцию.

5.2. Функции

Описание функции

Описание функции состоит из заголовка и тела функции и выглядит следующим образом:

<тип данных> <имя> (<параметры функции>)
{
< тело функции >
}

Первая строка представляет заголовок функции:

- Любая функция имеет тип. Функция может возвращать значение, тип которого совпадает с типом самой функции.
- Если функция не возвращает никакого значения, то она должна иметь тип **void** (такие функции иногда называют процедурами)
- При объявлении функции, после ее типа должно находиться имя функции, после которого в круглых скобках указываются параметры (аргументы) функции. Для каждого параметра указывается тип и имя. Параметр функции – это значение, которое можно передать функции при ее вызове. Если аргумент функции не один, а их несколько, то их нужно разделять запятой. Если аргументов в функции нет, то в скобках можно указать тип **void**.

После открывающей фигурной скобки идет тело функции, которое представляет собой последовательность описаний и операторов, которая начнет работать при вызове функции.

Для того чтобы вернуть какое-то значение из функции или вообще прекратить ее работу, необходимо использовать оператор return.

Вызов функции

Обращение к функции называют *вызовом*. Для вызова функции необходимо указать имя функции и в круглых скобках фактические параметры (если они есть):

<имя функции> (<фактические параметры>);

*Простая **void**-функция*

void (или по-русски пустота) – это специальный тип данных не принимающий никаких значений. В случае с **void**-функцией, тип **void** указывает на то, что такая функция ничего не возвращает программе.

Примером **void-**функций являются уже хорошо знакомые функции **printf( )** и **scanf( )**.

При объявлении функций в Си/Си++ не требуется никаких специальных ключевых слов. Пример создания простой **void**-функции приведен в таблице 5.1.

Таблица 5.1. Объявление простой void-функции

Код	Комментарии
void simple_functioin(void) {	// начало объявления функции
int local_value = 8; printf("%d\n", local_value);	
}	// конец функции

У любой функции могут быть собственные переменные. Такие переменные называются локальными. Доступ к таким переменным возможен только внутри самой функции.

В программировании *локальной* переменной называют переменную, объявленную внутри блока кода. Область видимости локальной переменной начинается в точке ее объявления и заканчивается в конце этого блока.

В программировании *глобальной* переменной называют переменную, областью видимости которой является вся программа.

В таблице 5.2 приводится пример описания глобальных и локальных переменных.

Таблица 5.2. Пример работы с глобальными и локальными переменными

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение библиотек
int value = 5; float float_value = 7.5;	// глобальные переменные

void simple_function(void)
{
int simple_value = 8;
value = 6;
}
	// простая void-функция

int main(void)
{
int main_value = 7;
	// основная функция

Код	Комментарии

//printf("%d\n", simple_value);
printf("%d\n", main_value);
printf("%.1f\n", float_value);
 	
printf("%d\n", value);
simple_function();
printf("%d", value);
	// запрещено
	// ответ: 7
	// ответ: 7.5
	
	// ответ: 5
	
	// ответ: 6

fflush(stdin); getchar();	// пауза
return 0; }	// завершение программы

Как видно из примера (таблица 5.2), простая **void**-функция вызывается следующим образом:

- имя функции;
- открывающаяся и закрывающаяся круглые скобки: ();
- точка с запятой.

***Void**-функция с параметрами*

Void-функция, рассмотренная в примере выше, имеет очень ограниченную область применения. Однако как уже упоминалось выше, сами по себе **void**-функции широко применяются в программировании, хорошим примером тому служит функция **printf()**. Отличает эту функцию возможность использования параметров.

При объявлении **void**-функции с параметрами, в круглых скобках осуществляется перечисление всех параметров функции (таблица 5.3).

Следует особо отметить, что если в качестве аргументов функции используются простые типы данных, то все изменения аргументов, произведенные внутри функции, никак не отразятся, на этих аргументы вне функции.

Таблица 5.3. Пример void-функции с параметрами

Код	Комментарии
void complex_functioin(int x, int y) {	// начало // объявления функции
int answer = x + y; printf("%d\n", answer);	
}	// конец функции

Обратите внимание, что перечисление параметров одного типа через запятую, при создании функции невозможно. В таблице 5.4 приведены примеры корректного и некорректного объявления параметров функции.

Таблица 5.4. Примеры корректного и некорректного объявления параметров функции

Код	Комментарии

void firs_functioin(int x, int y)
{
  printf("%d %d\n", x, y);
}
	// правильно

void firs_functioin(int x, y)
{
  printf("%d %d\n", x, y);
}
	// неправильно !!!

Массив в качестве параметра функции

Для передачи переменной сложного типа в качестве параметра функции используется символ «звездочка», который указывает на то, что мы используем особый тип данных – указатель на область памяти. Подробно об указателях будет рассказано в главе 6.

Следует помнить, что при передаче массива в качестве параметра функции любые изменения в массиве внутри функции приведут к изменениям в массиве вне функции. В таблице 5.5 приведен пример использования функции, одним из параметров которой является одномерный массив.

Таблица 5.5. Массив в качестве параметра функции

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение // библиотек

void complex_function(int x, int *array)
{
  x = 7;
  array[0] = 7;
}
	// void-функция

Код	Комментарии

int main(void)
{
  int i, x= 5;
  int array[5] = {0, 1, 2, 3, 4};
  complex_function(x, array);
	// основная
	// функция

for (i=0; i<5; i++)
  {
    printf("%d\n", array[i]);
  }
	
	// ответ: 7, 1, 2, 3, 4

printf("x = %d", x);	// ответ: 5
fflush(stdin); getchar();	// пауза
return 0; }	// завершение // программы

Функция, возвращающая значение

Помимо процедур (void-функций), в языке Си/Си++ возможно создание и классических функций, способных возвратить значение. Примерами таких функций являются: sin(), cos(), abs() и др.

Возврат значения осуществляется с помощью ключевого оператора **return** (таблица 5.6).

Таблица 5.6. Пример классической функции возвращающей значение

Код	Комментарии
int classic_functioin(int x, int y) {	// начало // объявления функции
int answer = x + y;	// локальная переменная
return answer;	// возврат значения
}	// конец функции

Инструкцию **return** не обязательно ставить в конце функции, ее можно поставить одновременно в нескольких местах, главное следить, чтобы в любом случае она была выполнена до завершения функции (до фигурной скобки).

В таблице 5.7 приведен пример, в котором используется одновременно несколько инструкций **return**.

Таблица 5.7. Пример корректного возврата значения функции

Код	Комментарии
int complex_function(int x, int y) {	// начало // объявления функции
int answer = x + y;	// локальная переменная

switch (answer)
{
case 0: return answer + 5;
case 1: return answer + 3;
case 2: return answer + 1;
 	
//default: return answer;
}
	// ветвление
	
	// корректный вариант №1

// return answer;	// корректный вариант №2
}	// конец функции

5.3. Рекурсивные функции

Рекурсия – это способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих ее операторов обращается сама к себе.

Таким образом, рекурсивная функция определяет свое значение, обращаясь к себе самой с другими аргументами. Если функция (метод) вызывает сама себя, то такой вызов называется *рекурсивным вызовом* функции.

Существует два типа рекурсий: **прямая** и **косвенная**. Их отличие друг от друга состоит в том, что при прямой рекурсии функция вызывает саму себя, тогда как при косвенной рекурсии функция обращается к себе опосредовано, путем вызова другой функции, в которой содержится обращение к первой.

Язык Си/Си++ допускает использование рекурсивных функций. Особенность рекурсивного описания функции состоит в том, что в теле такой функции содержится обращение к этой же описываемой функции. Следует отметить, что организация рекурсивных функций подчиняется строгому правилу: хотя бы одна из ветвей рекурсивной функции не должна содержать рекурсивного вызова. Эта ветвь определяет остановку процесса рекурсивного вызова. В противном случае

произойдет рекурсивное зацикливание с последующим аварийным завершением программы.

В качестве примера рассмотрим задачу нахождения факториала.

Напомним, что факториалом числа n называют произведение первых n натуральных чисел:

$$f(n) = n! = \begin{cases} 1 & \text{если } n = 1; \\ 1 \cdot 2 \cdot \ldots \cdot n & \text{если } n > 1. \end{cases}$$

Существует два способа решения этой задачи:

- с помощью цикла со счетчиком;
- с помощью рекурсии.

В таблице 5.8 приведена функция, которая находит факториал с помощью цикла.

Таблица 5.8. Прим ер итерационного вычисления факториала

Код	Комментарии
#include <stdio.h> #include "stdafx.h"	// подключение библиотек

int for_factorial(int n)
{
  int i,ans = 1;
  for (i = 2; i <= n; i++)
  {
    ans *= i;
  }
  return ans;
}
	// функция нахождения
	// факториала, через цикл for

int main(void)
{
  int f = for_factorial(5);
  printf("factorial(5) = %d", f);
  fflush(stdin);
  getchar();
  return 0;
}
	// основная функция
	
	// нахождение факториала
	// вывод результата

Реализация рекурсивного алгоритма предполагает реализацию рекурсивной функции. Рекурсивная форма записи функции факториал выглядит следующим образом:

$$\mathrm{f(n)} = \mathrm{n}! = \begin{cases} 1, & \text{если } \mathrm{n} = 1; \\ \mathrm{n} \cdot (\mathrm{n} - 1)!, & \text{если } \mathrm{n} > 1. \end{cases}$$

Чтобы циклический процесс преобразовать в рекурсивный, нужно определить три важных момента:

1. Формулу следующего элемента, который используется в рекурсивном процессе. Эта формула указывается в операторе **return**.

В рассматриваемом примере эта формула выглядит следующим образом:

$n! = n \cdot (n-1)!,$ если $n > 1$.

2. Список параметров, которые передаются в рекурсивную функцию. Один из параметров обязательно есть счетчик изменяющий свое значение. Другие параметры являются дополнительными. В нашем случае функция факториал имеет только один параметр n, который на каждой итерации принимает новое значение от n до 1.

3. Условие прекращения последовательности рекурсивных вызовов функции. Условие прекращения указывается в операторе **return**. При вычислении факториала таким условием будет $1! = 1$, если $n = 1$.

В таблице 5.9 приведена программа решения задачи вычисления факториала с использованием рекурсивной функции.

Таблица 5.9. Пример рекурсивного вычисления факториала

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение // библиотек
int factorial(int n) {	// функция «факториал»

if (n>1) return n*factorial(n-1);
  else   return 1;
}
	// рекурсивный
	// алгоритм

int main(void) {	// функция «main»
printf("x = %d", factorial(5));	// вывод результата (5!=120)
fflush(stdin); getchar();	// пауза
return 0; }	// завершение // программы

При решении многих математических задач описание рекурсивных функций обычно короче и нагляднее нежели, чем описание не рекурсивных функций. Однако вычисление рекурсивной функции является одной из наиболее ресурсозатратных операций для компьютера, так как требуется больше машинного времени (за счет повторных

обращений к функции) и памяти ЭВМ (за счет дублирования локализованных в функции переменных). Так же следует помнить, что глубина рекурсии в большинстве компиляторов ограничена. Поэтому при выборе способа описания функции следует решить, чему отдать предпочтение – эффективности программы или ее наглядности.

5.4. Внешние модули, библиотеки

Когда речь идет о достаточно больших проектах, наборы функций часто оформляют в виде отдельных модулей (файлов исходного кода), которые подключаются к программе, как библиотеки, с помощью директивы **#include**.

При создании модулей следует учитывать следующие особенности:

- Модуль на языке Си/Си++ является отдельным файлом с расширением *.h;
- Порядок, в котором вы перечисляете модули с помощью директивы **#include** часто оказывается важен, т.к. последующий модуль может использовать какие-то возможности предыдущего.
- Внутри модуля рекомендуется создавать функции, а также по необходимости директивы.
- Глобальные переменные в отдельных модулях никак не связаны с таковыми в любых других модулях (в том числе и в основном), однако при необходимости можно выполнить такую связь, в простейшем случае это делается с помощью ключевого слова **extern** (данный вопрос рекомендуется изучить самостоятельно, например, обратившись к [6]).

Вопросы для самопроверки

1. Что такое подпрограмма?
2. Что такое функция?
3. Опишите тип данных **void**.
4. Приведите пример **void**-функции.
5. Необходимо определить в таблице 5.2:
6. Какие переменные являются глобальными и локальными?
7. К каким переменным можно обратиться из функции main?
8. Как осуществляется передача массива в качестве параметра функции?
9. Приведите пример классической функции.

Практические задания

1. Получить таблицу значений функции $y = 2x^2 + x+-4$ при x, изменяющемся от -5 до 5 с шагом 1. Вычисление значений функции оформить в виде функции. Результаты представить в виде таблицы.

2. Получить таблицу значений функции $y = sh(x)$ при x, изменяющемся от -1 до 1 с шагом 0.1. Вычисление значений функции оформить в виде функции. Результаты представить в виде таблицы. Функция гиперболический синус определяется формулой $sh = \frac{e^x - e^{-x}}{2}$.

3. Два треугольника заданы длинами своих сторон a, b и c. Вычислить площади треугольников по формуле Герона и определить наибольший из них.

4. Футболист ударом ноги посылает мяч вертикально вверх с высоты 1 м с начальной скоростью 20 м/с. На какой высоте мяч будет через 1, 3, 4 с? Движение мяча описывается зависимостью:

$$y(t) = y_0 + v_0 t + \frac{gt^2}{2},$$

где y_0 – начальная высота; v_0 – начальная скорость; t – время. Оформить вычисление $y(t)$ в виде функции.

Практическая работа № 7. Программирование с использованием функций пользователя

Цель работы – овладение практическими навыками алгоритмизации и программирования задач с использованием функций пользователя.

Задание к работе

В соответствии с вариантом задания (таблица 5.10), в среде Visual Studio необходимо разработать программу, выполняющую:

- ввод данных из консоли и, при необходимости, генерацию случайных данных;
- расчет результатов с применением функции пользователя;
- вывод в консоль исходных данных и результатов расчета.

В графе «Примечание» таблицы 5.10 указан тот алгоритм, который следует оформить в виде функции.

Более подробно, типовые алгоритмы, рассматриваемые в настоящей практической работе, описаны в [6, 7].

Таблица 5.10. Варианты заданий к практической работе № 7

№ вар.	Формулировка задания	Примечание
1	Даны три квадратные матрицы a[5,5], b[7,7], c[10,10]. Найти длину вектора $\overline{x} = (x_1, x_2, x_3)$, где x_1, x_2, x_3 – суммы элементов матриц a, b, c.	Вычисление суммы элементов матрицы.
2	Решить уравнение $d \cdot x = c$, где d – длина вектора, $\overline{a} = (a_1, ..., a_n)$, $n \le 7$; c – длина вектора $b = (b_1, ..., b_m)$, $m \le 9$.	Вычисление длины вектора.
3	Даны два вектора $\overline{x} = (x_1, ..., x_n)$ и $\overline{y} = (y_1, ..., y_n)$, $n \le 9$. Найти угол между векторами x и y по формуле: $\alpha = \arccos\left(\frac{x_1 \cdot y_1 + x_2 \cdot y_2 + ... + x_n \cdot y_n}{\sqrt{x_1^2 + x_2^2 + ... + x_n^2} \cdot \sqrt{y_1^2 + y_2^2 + ... + y_n^2}}\right)$.	Вычисление скалярного произведения.
4	Четыре точки заданы своими координатами $x(x_1, x_2)$, $y(y_1, y_2)$, $z(z_1, z_2)$, $p(p_1, p_2)$. Выяснить, какие из них находятся на максимальном расстоянии друг от друга.	Вычисление расстояния.
5	Заданы три вектора $\overline{x} = (x_1, ..., x_n)$, $\overline{y} = (y_1, ..., y_m)$, $\overline{z} = (z_1, ..., z_l)$, $n \le 7$, $m \le 9$, $l \le 5$. Упорядочить по возрастанию минимальные элементы каждого их этих векторов.	Вычисление минимального элемента.
6	Заданы три матрицы a[3,4], b[4,5], c[6,3]. Вычислить след каждой матрицы и найти наименьший из них. (Следом матрицы называется сумма элементов главной диагонали).	Вычисление следа матрицы.
7	Заданы длины сторон двух треугольников *ABC* и *PLE*. Найти сумму и разность площадей этих треугольников. Площадь треугольника *MNK* вычисляется по формуле Герона: $s = \sqrt{p \cdot (p - m) \cdot (p - n) \cdot (p - k)}$, где m, n, k – длины сторон треугольника, p – полупериметр треугольника.	Вычисление площади треугольника.

№ вар.	Формулировка задания	Примечание
8	Задан массив чисел $\overline{a} = (a_1, ..., a_n)$, $n \le 11$. Выбрать из него числа, принадлежащие отрезку $[x,\ y]$. Из выбранных чисел образовать вектор.	Проверка на принадлежность отрезку.
9	Три точки заданы своими координатами $x(x_1,\ x_2)$, $y(y_1,\ y_2)$, $z(z_1,\ z_2)$. Вывести в консоль координаты этих точек по возрастанию угла между осью абсцисс (x) и лучом, соединяющим начало координат с соответствующей точкой.	Вычисление угла.
10	Вычислить z – суммы значений функции $z = f(a,b) + f(a-1,b) + f(a-b,b) + f(a,b-1)$, где: $f(u,t) = \begin{cases} u+t & \textit{если} \quad u \ge 0, t \ge 0; \\ u-t & \textit{если} \quad u < 0, t < 0; \\ u+t & \textit{в остальных случаях.} \end{cases}$	Вычисление $f(u,t)$.
11	Вычислить z – суммы значений функции $z = f(\cos(a), b) + f(\cos(a), b) + f(\sin(a), b-1) +$ $+ f(\sin(a) - 1, \cos(a) + b),$ где: $f(u,t) = \begin{cases} u + \sin(t) & \textit{если} \quad u > t; \\ u+t & \textit{если} \quad u \le t. \end{cases}$	Вычисление $f(u,t)$.
12	Вычислить z – суммы значений функции $z = f(\sqrt{\lvert x \rvert}, y) + f(\sqrt{\lvert x \rvert} + 1, -y) + f(\lvert x \rvert - \lvert y \rvert, x)$, где: $f(u,t) = \begin{cases} u + 2 \cdot t & \textit{если} \quad u \ge 0; \\ u+t & \textit{если} \quad u \le -1; \\ u - 2 \cdot t + 1 & \textit{если} \quad -1 < u < 0. \end{cases}$	Вычисление $f(u,t)$.
13	Даны три матрицы a[5,5], b[4,4], c[3,3]. Найти длину вектора $\overline{x} = (x_1, x_2, x_3)$, где: x_1 – количество положительных элементов матрицы a; x_2 – количество отрицательных элементов матрицы b; x_3 – количество нулевых элементов матрицы c.	Вычисление количества элементов заданного типа.

№ вар.	Формулировка задания	Примечание
14	Решить уравнение $d \cdot x = c$, где d – наименьший элемент вектора $\overline{a} = (a_1, ..., a_n)$, $n \le 7$; c – наибольший элемент вектора $b = (b_1, ..., b_m)$, $m \le 9$.	Нахождение наименьшего и наибольшего элементов массива.
15	Заданы два вектора: $\overline{x} = (x_1, ..., x_n)$ и $\overline{y} = (y_1, ..., y_n)$, где $n \le 9$. Вычислить полусуммы длин данных векторов.	Вычисление длины вектора.
16	Три точки заданы своими координатами $x(x_1, x_2)$, $y(y_1, y_2)$, $z(z_1, z_2)$. Вычислить, какие их них находятся на минимальном расстоянии друг от друга.	Вычисление расстояния.
17	Даны три вектора $\overline{x} = (x_1, ..., x_n)$, $\overline{y} = (y_1, ..., y_n)$, $\overline{z} = (z_1, ..., z_n)$, $n \le 10$. Упорядочить по возрастанию максимальные элементы этих векторов.	Нахождение наибольшего элемента в массиве.
18	Заданы три матрицы a[3,4], b[5,3], c[4,4]. Вычислить след каждой матрицы и найти наибольший из них. (Следом матрицы называется сумма элементов главной диагонали).	Вычисление следа матрицы.
19	Заданы длины сторон двух треугольников *ABC*, *IJK* и *PLE*. Вычислить площади этих треугольников и найти наибольшую из них. Площадь треугольника *MNK* вычисляется по формуле Герона: $s = \sqrt{p \cdot (p-m) \cdot (p-n) \cdot (p-k)}$, где m, n, k – длины сторон треугольника, p – полупериметр треугольника.	Вычисление площади треугольника.
20	Заданы два вектора $\overline{a} = (a_1, ..., a_n)$, $n \le 11$; и $b = (b_1, ..., b_m)$, $m \le 17$. Выбрать из них числа, принадлежащие отрезку $[x, y]$ и из выбранных чисел сформировать вектор c.	Проверка принадлежности отрезку.

Порядок выполнения работы

1. Изучить:
- правила записи функций и обращения к ним;
- способы передачи параметров в функцию;
- порядок выполнения программ, использующих функции.

2. Разработать алгоритм решения задачи в соответствии с заданием.

3. Составить программу решения задачи. Программа должна включать в себя четыре составляющие: функция пользователя, блок ввода данных, блок обработки данных (с применением функции пользователя) и блок вывода результатов обработки.

4. Набрать программу в редакторе системы программирования Microsoft Visual Studio C++.

5. Протестировать разработанную программу. При наличии ошибок внести изменения в программу.

6. Оформить отчет к работе. Отчет должен содержать:

- цель работы;
- задание к работе (с указанием варианта);
- описание алгоритма;
- исходный код программы на языке Си/Си++ (с комментариями);
- результаты тестирования (в том числе скриншот работы программы);
- выводы.

Контрольные вопросы и задания

1. В каком случае возникает необходимость использования функций пользователя?
2. В чем отличие функций от процедур? (В использовании, в оформлении, в обращении)?
3. Укажите способы передачи параметров в функцию.
4. Укажите способы получения результатов работы функции.
5. Сформулируйте правила обращения с функциями.

Глава 6. РАБОТА С УКАЗАТЕЛЯМИ И ФАЙЛАМИ

6.1. Работа с указателями

Указатель, как особый тип данных

Указателем называется переменная, которая содержит значение адреса элемента памяти, где хранится значение некоторого объекта. То есть указатель ссылается на блок данных из области памяти, причем на самое его начало. Указатель может ссылаться на переменную или функцию. Если значение переменной value хранится в элементе, расположенном по адресу 0xFF00FF11, то и указатель на переменную value будет иметь значение *0xFF00FF11.

При использовании в программе имени того или иного объекта говорят о прямом доступе к объекту. В том случае, когда адрес объекта помещен в указатель, то речь идет о косвенном доступе к объекту, на который «смотрит» указатель.

В основном указатели используются для организации динамического распределения памяти, например при объявлении массива, не надо будет его ограничивать в размере.

Любой указатель, как и любую переменную, необходимо объявить перед использованием.

Описание указателя имеет следующий общий вид:

тип данных *имя указателя;

Оно ничем не отличается от описания простой переменной, единственным отличием является использование символа «*» (звездочка), для указания на то, что мы создаем не простую переменную, а указатель (таблица 6.1).

Таблица 6.1. Пример объявления указателя

Код	Комментарии
int value;	// объявление простой переменной
int *pointer;	// объявление указателя на объект типа int

По такому принципу можно создавать указатели на объекты любого простого типа данных (**int, float, char, bool** и др.)

Работа с указателями на объекты простых типов данных

Непосредственная работа с памятью по жестко заданному адресу, как правило, не предполагается, вместо этого указателю обычно присваивается адрес какой-то существующей переменной или области памяти.

Возникает вопрос – как присвоить указателю адрес объекта? Для этого нужно знать адрес переменной или функции, который можно получить с помощью операции взятия адреса «&» (символ амперсант). Присвоение адреса производится оператором присвоения «=».

В таблице 6.2 приводится пример создания указателя, присвоение ему значения и вывод результата на экран. Обратите особое внимание на то, где используется (или не используется) звездочка, а где амперсант.

Таблица 6.2. Пример работы с простым указателем на int-переменную

Код	Комментарии
int value;	// объявление простой переменной
int *pointer;	// объявление указателя на объект тип int
pointer = &value;	// присваиваем указателю pointer // адрес переменной value
scanf("%d", pointer);	// считываем данные по адресу указателя
printf("%d", value);	// выводим результат

Обратите особое внимание, что изначально указатель ни на что не указывает! Прежде чем задать указатель обязательно требуется создать переменную, на которую и будет указывать этот указатель.

Поменять значение переменной, хранящейся по адресу, который записан в указателе, можно с помощью указателя. Для этого в операторе присваивания перед именем указателя ставится символ звездочка (таблица 6.3).

Таблица 6.3. Пример изменения данных с применением указателя

Код	Комментарии
int value;	// объявление простой переменной
int *pointer;	// объявление указателя на объект типа int
pointer = &value;	// присваиваем указателю pointer // адрес переменной value

*pointer = 25;	// присвоение значения по адресу,
	// хранимому в указателе pointer,
	// значение переменной value равно 25

printf("%d", value);	// выводим результат

Указатели используются не только при работе с переменными. Благодаря указателям, можно производить изменение аргументов функций. Мы уже имели дело с таким использованием указателей, когда рассматривали функцию **scanf**().

В таблице 6.4 приводится пример использования указателей при описании функций.

Таблица 6.4. Пример использования указателя при описании функций

Код	Комментарии
void fnc(int first, int *second) {	// пользовательская функция

first = 5;
 	
*second = 7;
}
	// не изменится вне функции
	
	// изменится после выполнения
	// функции

int main(void) {	// основная функция
int first = 1, second = 1; int *pointer = &second;	 // Указатель pointer указывает на переменную second.
fnc(first, pointer);	// выполнение функции
printf("%d %d", first, second);	// вывод результата: «1 7»
fflush(stdin); getchar();	// пауза
return 0; }	// завершение программы

Указатель на массив

Указатель может указывать не только на простые переменные, но и на переменные сложных типов данных, например массивов.

Если некоторому указателю присваивается имя массива или его адрес, то это эквивалентно записи в указатель адреса первого элемента этого массива, т.е. элемента с индексом 0.

Когда указатель «смотрит» на начало массива для обращения к любому элементу массива можно воспользоваться разными способами (таблица 6.5).

Таблица 6.5. Обращения к элементу массива по указателю

Код	Комментарии
int array[3] = {1,2,3};	// объявление массива
int *pointer=array;	// указатель pointer смотрит на array[0]
pointer [1] = 5;	// теперь array[1] равен 5
*(array+2) = 10;	// теперь array[2] равен 10

Разберемся, что означает последняя строка кода.

В Си/Си++ конструкция a[b] означает не что иное, как *(a+b), где a – объект, b – смещение от начала памяти, адресующей объект. Таким образом, обращение к элементу массива a[i] может быть записано и как *(a+i),

Подобный способ обращения к массивам используется, например, при упаковке данных из динамической памяти.

Следует обратить внимание на то, что квадратные скобки используются не только при обращении к элементам массива, но и при создании массива. Если использование квадратных скобок при обращении к элементам массива, как мы выяснили, не является обязательным, то при создании обычного массива обойтись без квадратных скобок не получится.

Указатель на экземпляр структуру

При написании функций часто требуется изменить значение элементов экземпляра структуры передаваемого в качестве аргумента функции, для этого требуется вместо самого экземпляра структуры передать функции указатель на этот экземпляр. Особенность состоит

в том, что во время работе с указателями на экземпляры структуры, при обращении к их элементам, символ звездочка не применяется, вместо этого используется «стрелка» состоящая из дефиса и символа «больше». В таблице 6.6 приведен программный код, иллюстрирующий работу с указателем на экземпляр структуры.

Таблица 6.6. Пример работу с указателем на экземпляр структуры

Код	Комментарии

struct my_struct
{
  int value;
};
	// структура с одним
	// элементом

void fnc(struct my_struct *st) {	// пользовательская функция
st->value = 5; }	// присвоение значения // элементу экземпляра структуры
int main(void) {	// основная функция
struct my_struct st;	// объявление переменных
st.value = 1;	// присвоение значения // элементу экземпляра структуры
fnc(&st);	// выполнение функции
printf("%d", st.value);	// вывод результата (число 5)
fflush(stdin); getchar();	// пауза
return 0; }	// завершение программы

Динамическая память, работа с областями памяти

При работе с указателями осуществляется непосредственная работа с памятью (как правило, с ОЗУ). Возникает вопрос: можно ли самостоятельно выделить в оперативной памяти какую-то область памяти, а затем ссылаться к ней с помощью указателя, при этом, не создавая никаких переменных? Ответ – да можно, с помощью специальных функций библиотеки **stdlib.h**. Рассмотрим некоторые часто используемые функции этой библиотеки.

Функция **malloc()** (memory allocate – выделение блока памяти). Данная функция имеет один аргумент – объем памяти (в байтах), который необходимо выделить и возвращает указатель на начало этой

области памяти. Пример использования функции **malloc**() приведен в таблице 6.7.

Таблица 6.7. Пример использования функции malloc

Код	Комментарии
#include <stdlib.h> …	// библиотека, необходимая для // вызова функции malloc
int *pointer;	// объявление указатель // на элемент памяти
pointer = (int*)malloc(40);	// выделение памяти // под 10 элементов int (40 байт)
pointer[9] = 7;	// обращение к памяти

Для освобождения ранее выделенной области динамической памяти используется функция **free()**. Данная функция принимает один аргумент – указатель любого типа и не возвращает никаких значений. В таблице 6.8 приведен пример использования функции **free**().

Таблица 6.8. Пример использования функции free()

Код	Комментарии
int *pointer;	// объявление указателя // на элемент памяти
pointer = (int*)malloc(40);	// выделение памяти // под 10 элементов int
free(pointer);	// освобождение памяти

Упаковка данных из динамической памяти

При написании программ часто возникает необходимость передать данные на удаленное устройство, например через интернет. Для этого требуется передавать информацию побайтно. Но что делать, есть нам необходимо побайтно передать (например, через Internet) массив типа **int**, каждый из элементов которого занимает 4 байта? Ответ – воспользоваться указателями и переопределением типов данных. В таблице 6.9 представлен пример упаковки данных. Суть примера состоит в побайтной передаче данных из объекта (в данном примере – массива) любого сложного типа.

Таблица 6.9. Пример упаковки данных

Код	Комментарии
int i; int array[5] = {1,2,3,4,5}; char *pointer;	// объявление переменных

pointer = (char*)array;
for (i=0; i<5*4; i++)
{
  send(*(pointer +i));
	// отправка данных,

}	// завершение передачи

Вопросы для самопроверки

1. Что такое указатель?
2. Как указатели позволяют расширить возможности функций?
3. Является ли массив указателем?
4. Как создать динамический массив?
5. Что такое упаковка данных и для чего она применяется?

Практические задания

1. Необходимо написать программу на языке Си, которая создает массив из 10 элементов, заполняет его числами, находит сумму этих чисел, а затем выводит результат на экран. При решении этой задачи, использовать квадратные скобки [] запрещается!

2. Необходимо написать программу на языке Си, которая создает массив arr из 10 элементов, находит в нем элемент arr[index] и выводит результат на экран. Процедура поиска должна быть оформлена в виде функции, которая возвращает результат в виде указателя.

3. Необходимо написать программу на языке Си, которая создает структуру содержащую минимум 4 поля разных типов (например, char, short, int, unsigned int), затем выполнить упаковку данных указанной структуры и вывести ее содержимое на экран в виде массива однобайтных шестнадцатеричных чисел (типа unsigned char).

6.2. Работа с файлами

При создании переменной в Visual Studio C++ мы создаем ее в оперативном запоминающим устройстве (ОЗУ), соответственно, после завершения программы и/или выключении питания компьютера эти данные исчезают.

Наиболее простым способом сохранения данных после завершения работы программы является использование файлов. Данные, содержащиеся в файлах, могут иметь самый разнообразный характер: исходные данные для работы программ или результаты выполнения программ; произвольные тексты; графические изображения и т.п.

Файл – это поименованная область внешней памяти, содержащая некоторые данные. Следовательно, под файлом понимается некоторая последовательность байтов внешней памяти компьютера, которая имеет своё, уникальное имя, например, D:\Apps\FP1.dat.

Имя файла, под которым он фигурирует в операционной системе, называется физическим именем. Но это взгляд на файлы с точки зрения операционной системы компьютера. А что означает файл для программы, написанной на языке Си/Си++? В программах, написанных на языке Си/Си++, файл рассматривается как структура данных. Имя файловой структуры принято называть логическим именем файла.

Различают два вида файлов: *текстовые* и *бинарные*.

Текстовые файлы представляют собой последовательность ASCII символов и могут быть просмотрены и отредактированы с помощью любого текстового редактора. Эта последовательность символов разбивается на строки символов, при этом каждая строка заканчивается двумя кодами «возврат каретки» (0x0D или «/r») и «перевод строки» (0x0A или «/n»).

Бинарные (двоичные) файлы представляют собой последовательность данных, структура которых определяется программно.

В языке Си/Си++ не предусмотрены никакие заранее определенные структуры файлов. Все файлы рассматриваются компилятором как последовательность (поток байт) информации.

Для файлов определен указатель чтения-записи данных, который определяет текущую позицию доступа к файлу.

Основными операциями с файлами является чтение данных из файла и запись данных в файл. В языке Си/Си++имеется большой набор функций для работы с файлами, большинство которых находятся в библиотеке **stdio.h.** Основные функции по работе с файлами представлены на рисунке 6.1.

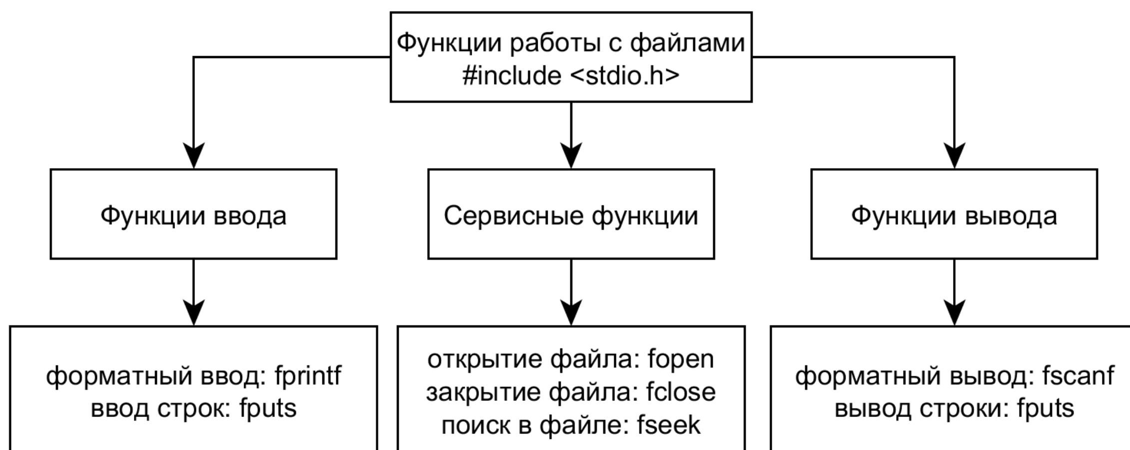

Рисунок 6.1 – Основные функции по работе с файлами на языке Си/Си++

Открытие файла

Как уже отмечалось выше, файлу в программе присваивается внутреннее логическое имя, используемое в дальнейшем при обращении к нему. Логическое имя – это указатель на файл, т.е. на область памяти, где содержится вся необходимая информация о файле.

Формат объявления указателя на файл следующий:

FILE *ID_указателя_на_файл;

Для того чтобы иметь возможность чтения или записи данных в файл, его нужно открыть для доступа.

Для того чтобы открыть файл, необходимо выполнить функцию **fopen**(char * ID_файла, char *режим)

указав два параметра:

- физическое имя файла на диске;
- режим доступа (массив char, заканчивающийся символом 0x00).

Физическое имя файла, т.е. ID_файла, задается строкой, которая содержит полное имя файла, например, "C:\\data\\file.txt". Если указать только имя файла без пути к нему, то при этом программа считает, что этот файл находится в текущей папке.

Следует обратить особое внимание на два обратных слеша в физическом имени файла!

Второй параметр функции **fopen()** – строка, в которой задается режим доступа к файлу.

Возможные значения данного параметра следующие:

w – файл открывается для записи (write); если файла с заданным именем нет, то он будет создан; если же такой файл уже существует, то перед открытием прежняя информация уничтожается;

r – файл открывается для чтения (read); если такого файла нет, то возникает ошибка;

a – файл открывается для добавления (append) новой информации в конец;

r+ (*w*+) – файл открывается для редактирования данных, т.е. возможны и запись, и чтение информации;

a+ – то же, что и для *a*, только запись можно выполнять в любое место файла (доступно и чтение файла);

t – файл открывается в текстовом режиме;

b – файл открывается в двоичном режиме;

Данная функция берет физическое имя файла на внешнем носителе и ставит ему в соответствие логическое имя (указатель файла).

При успешном открытии файла функция **fopen()** возвратит указатель на файл. В противном случае возвращается значение 0, что будет означать, что файла либо не существует, либо доступ к нему ограничен.

Чтение данных из файлов

Процедура чтения данных из файла состоит из трех основных операций:

- открытие файла для чтения;
- поиск нужной позиции в файле;
- чтение данных;
- закрытие файла.

Открытие файла для чтения осуществляется с функции **fopen()**, в которой второй параметр имеет значение "r" (в двойных кавычках). В табл. 6.10 приведен пример открытия файла.

Таблица 6.10. Открытие файла для чтения

Код	Комментарии
FILE *f = fopen("C:\\data\\file.txt", "r");	// открытие // файла

После того, как файл открыт и вы убедились, что указать "f" не нулевой, можно переходить к поиску нужной позиции в файле.

Для поиска нужной позиции в файле используется команда **fseek()** (поиск по файлу).

Функция **fseek()** позволяет обрабатывать файл подобно массиву и непосредственно достигать любого определенного байта в файле, открытом функцией **fopen()**. Функция **fseek()** имеет три аргумента:

- **filestream –** указатель на объект типа FILE;
- **offset** – количество байт для смещения, относительно некоторого положения указателя.
- **origin** – позиция указателя, относительно которой будет выполняться смещение. Такая позиция задatтся одной из следующих констант:

SEEK_SET – начало файла;
SEEK_CUR – текущее положение в файле;
SEEK_END – конец файла.

Функция **fseek( )** возвращает значение типа **int.** В случае успеха, функция возвращает нулевое значение. В противном случае, она возвращает ненулевое значение.

Пример использования этой команды приводится в таблице 6.11.

Таблица 6.11. Поиск нужной позиции в файле

Код	Комментарии
fseek(f, 6, SEEK_SET);	// перемещение // на 6-ю позицию

После перемещения на нужную позицию можно выполнить чтение массива данных из ПЗУ, для этого проще всего воспользоваться командой **fgets**() (get string from file или чтение строки из файла).

Функция **fgets()** имеет три аргумента:

- строка (массива char), в которую необходимо считать данные;
- количество байт, которое необходимо считать;
- указатель на файл.

Функция возвращает истинное значение, если чтение успешно выполнено и ложное, если прочитать данные не удалось. Пример работы функции **fgets()** приводится в таблице 6.12.

Таблица 6.12. Пример чтения данных из файла с помощью функции fgets()

Код	Комментарии
char str[25]; bool state;	// объявление // переменных
state = fgets(str, 10, f);	// чтение данных

if (state == true)
  printf("OK");
else
  printf("Error!");
	// проверяем, удалось ли
	// прочитать данные из файла

Данные не удается прочитать, например, если достигнут конец файла, благодаря такой особенности нетрудно организовать чтение файла до самого конца (считывать данные до тех пор, пока state не станет равной false).

Обнаружить конец файла можно и другим способом. Файл всегда заканчивается специальным символом **EOF** (end of file – конец файла), если в потоке данных обнаруживается этот символ, то конец файла был достигнут. Функция **feof()** возвращает 1, если был достигнут конец файла; в противном случае она возвращает 0.

Если требуется форматное чтение данных из файла, рекомендуется воспользоваться функцией **fscanf()**. Синтаксис этой функции полностью соответствует функции **scanf()**, единственное отличие – первым аргументом следует поставить указатель на файл. В таблице 6.13 приводится пример использования функции **fscanf()**.

Таблица 6.13. Пример использования функции fscanf()

Код	Комментарии
char value;	// объявление переменных
fscanf(f, "%d", &value);	// чтение данных

Запись данных в файл

Открытие файла для записи осуществляется знакомой нам функцией **fopen(),** в которой второй параметр имеет значение "w" – write (запись) или, например "a" – append (добавление).

Следует особо отметить, что если файл еще не был создан, то после выполнения открытия файла для записи – производится его создание.

Создание файлов невозможно в некоторых директориях компьютера, в случае использования жесткого диска с файловой системой NTFS. Данная файловая система способна разграничить права доступа к определенным областям памяти. В случае использования диска с файловой системой FAT32 никаких разграничений прав доступа не предусмотрено, однако диск может быть полностью защищен от записи (такая возможность, есть например, у карт памяти формата SD).

Следует также отметить, что в случае работы с файлами имеются некоторые ограничения на выполнение одновременных операций, например нельзя осуществлять запись в файл, который уже открыт для записи другим приложением.

Как и в случае чтения, запись в файл можно выполнять с помощью функций вывода, рассмотренных в параграфе 2.6, с добавлением литеры f в начале. Наиболее распространенными среди них являются функции записи в файл **fprintf**() и **fpus**(). Пример использования функции **fprintf**() приводится в таблице 6.14.

Таблица 6.14. Открытие файла

Код	Комментарии
int value = 7;	// объявление переменных
fprintf(f, "%d", value);	// запись данных

Как и в случае с функциями чтения, при записи в файл, можно переместиться на нужную позицию, для этого нужно воспользоваться функцией **fseek()** однако следует помнить, что **fseek()** работает не во всех режимах записи. Рекомендуется использовать **fseek()**, например в режиме «r+».

Закрытие файлов

Закрытие файла обычно происходит автоматически в случае успешного завершения программы (после строки return 0 в main-функции). Однако программа не всегда завершается штатно, например, если программа пришла к бесконечному циклу, ее завершение происходит в нештатном режиме, в этом случае файлы не будут закрыты! При этом может произойти либо потеря данных, либо ограничение доступа к данным из других приложений.

Рекомендуется всегда закрывать открытые как для чтения, так и для записи файлы вручную в тот момент, когда дальнейшая работа с файлами становится нецелесообразной. Для этого существует специальная функция **fclose**() (file close – закрытие файла).

Данная функция имеет один аргумент – указатель на файл и возвращает результат закрытия файла. Если файл закрыть не удается (например, если файл перестал существовать), то функция **fclose**() возвращает значение отличное от нуля, иначе она возвращает ноль.

Мы изучили все основные функции работы с файлами, этого достаточно, чтобы написать свою программу которая, например, читает данные из файла, производит над ними определенные действия и записывает результат в новый файл. В таблице 6.15 приведена программа, которая считывает 10 значений типа **int** из файла с именем input.txt, суммирует их, а затем результат записывает в файл с именем output.txt. Считается, что файл input.txt существует и находится в корне диска «C:/».

Таблица 6.15. Пример обработки файлов на языке Си/Си++

Код	Комментарии
`#include "stdafx.h"` `#include <stdio.h>`	// подключение // библиотек
`int main(void)` `{`	// основная функция
`FILE *f_in, *f_out;`	// файловые указатели
`int value, i;` `int sum = 0;`	// объявление простых //переменных
`f_in = fopen("C:\\input.txt", "r");` `f_out = fopen("C:\\output.txt", "w");`	// открытие файлов

`for (i=0; i<10; i++)`
`{`
`fscanf(f_in, "%d", &value);`
`sum += value; }`
	// чтение и
	// обработка
	// данных файла

`fprintf(f_out, "answer = %d", sum);`	// запись данных
`fclose(f_in);` `fclose(f_out);`	// закрытие файлов
`return 0;` `}`	// завершение // программы

Вопросы для самопроверки

1. Какими особенностями обладает хранение данных в ПЗУ?
2. Как осуществляется открытие файлов на языке Си/Си++?
3. Как используется функция **fseek()**?
4. Какие функции ввода/вывода данных в файл языка Си/Си++ вы знаете?
5. Для чего требуется закрывать файл после завершения работы с ним?

Практические задания

1. Необходимо написать программу на языке Си, которая вводит с клавиатуры имя файла, а затем открывает его для чтения и выводит на экран.
2. Необходимо написать программу на языке Си, с помощью которой можно отредактировать текстовый файл, дописав в него строку либо удалив из него все данные.
3. Необходимо написать программу на языке Си, которая выводит «секретное сообщение», после того как пользователь введет верный пароль, при этом пароль хранится в текстовом файле.
4. Необходимо написать программу на языке Си, которая водит с клавиатуры предмет и оценку, а затем сохраняет результат в текстовый файл.
5. Необходимо написать программу на языке Си, которая открывает файл электронной зачетной книжки, полученной в ходе выполнения задания №4, а затем находит в ней предметы, сданные на «отлично» и выводит результат на экран.

Практическая работа № 8. Работа со структурами данных и файлами

Цель работы – овладение практическими навыками работы с структурами и файлами, а также особенностями их ввода и вывода.

Задание к работе

В соответствии с вариантом задания (таблица 6.16), в среде Visual Studio необходимо разработать программу выполняющую:

- формирование заданной структуры данных;

- ввод и вывод данных, в том числе:
    - ввод данных из консоли (в ОЗУ);
    - дополнение файла записями из ОЗУ;
    - загрузку данных из файла в ОЗУ;
    - вывод полной таблицы данных из ОЗУ в консоль;
    - удаление файла.
- расчет результатов с применением функций пользователя;
- вывод в консоль результатов расчета.

Программу рекомендуется оформить в виде бесконечного цикла, в котором у пользователя есть возможность выполнить одно из семи возможных действий:

- добавить запись в массив структур;
- вывести на экран массив структур (в виде таблицы);
- вывести на экран результаты обработки массива структур;
- сохранить массив структур в файл;
- загрузить массив структур из файла;
- удалить файл;
- выйти из программы.

Таблица 6.16. Варианты заданий к практической работе № 8

№ вар.	Имя объекта	Поля структуры	Задачи обработки
1	Студент	а) ФИО студента; б) группа; в) оценки по пяти экзаменам;	Распечатать данные о студентах отличниках.
2	Студент	а) ФИО студента; б) год рождения; в) место жительства (город или сельская местность).	Распечатать данные о студентах, проживающих в городе.
3	Книга по языку Си	а) ФИО автора; б) название; в) издательство; г) год выпуска.	Вывести в консоль литературу, изданную не ранее 2000 г.
4	Студент	а) ФИО студента; б) группа; в) оценка по пяти экзаменам.	Вывести в консоль данные о студентах, успевающих на «хорошо» и «отлично».

№ вар.	Имя объекта	Поля структуры	Задачи обработки
5	Абонент телефона	а) ФИО абонента; б) год регистрации номера; в) номер телефона.	Вывести в консоль данные об абонентах, у которых телефон зарегистрирован с **xxxx** года. Номер года вводится через консоль.
6	Книга по языку Си	а) ФИО автора; б) название; в) издательство; г) год выпуска.	Вывести в консоль данные о книгах, отсортировав их названия по алфавиту.
7	Студент	а) ФИО студента; б) группа; в) год рождения; г) оценки по пяти экзаменам.	Вывести в консоль данные о студентах, получивших оценку «удовлетворительно».
8	Студент	а) группа; б) ФИО студента; в) оценки по пяти экзаменам; г) отметки о пяти зачетах («з» – зачет, «н» – незачет).	Вывести в консоль фамилии неуспевающих студентов с указанием их группы и количества задолженностей.
9	Книга	а) ФИО автора; б) название; в) издательство; г) год выпуска.	Вывести в консоль книги одного издательства, название издательства вводится с клавиатуры.
10	Студент	а) ФИО студента; б) группа; в) оценки по четырем экзаменам.	Посчитать и вывести в консоль средний балл, полученный каждым студентом группы **x**. Название группы вводится с клавиатуры.
11	Рабочий	а) ФИО рабочего; б) номер цеха; в) размер заработной платы за месяц.	Посчитать и вывести в консоль общую суммы выплат по цеху **x**, а также среднемесячный заработок рабочего этого цеха. Номер цеха вводится с клавиатуры.
12	Книга	а) ФИО автора; б) название; в) издательство; г) год выпуска.	Вывести в консоль книги автора **x**. Имя автора вводится с клавиатуры.

№ вар.	Имя объекта	Поля структуры	Задачи обработки
13	Студент	а) ФИО студента; б) группа; в) оценки по пяти экзаменам.	Вывести в консоль данные о студентах, имеющих средний балл больше 3,5.
14	Студент	а) ФИО студента; б) группа; в) год окончания школы; г) место жительства.	Вывести в консоль данные о студентах, окончивших школу в **xxxx** году. Год требуется ввести с клавиатуры.
15	Книга	а) шифр книги; б) ФИО автора; в) название; г) год издания; д) номер стеллажа.	Вывести в консоль местоположение (номер стеллажа) книги автора **x** названия **y**. Значения **x** и **y** ввести с клавиатуры.
16	Пациент клиники	а) ФИО пациента; б) пол; в) возраст; г) место жительства; д) диагноз.	Вывести в консоль список иногородних, прибывших в клинику.
17	Студент	а) ФИО студента; б) год рождения; в) группа; г) место рождения.	Вывести в консоль данные о студентах родившихся в г. Владимире.
18	Книга	а) ФИО автора; б) название; в) издательство; г) год выпуска.	Вывести в консоль книги, изданные с 1994 по 2004 гг.
19	Пациент клиники	а) ФИО пациента; б) возраст; в) место жительства; г) диагноз.	Вывести в консоль список пациентов старше **x** лет с диагнозом **y**. Значения **x** и **y** ввести с клавиатуры.
20	Абонент телефона	а) ФИО абонента; б) год регистрации номера; в) название улицы, где живет абонент; г) номер телефона.	Вывести в консоль список абонентов, проживающих на улице **x**. Название улицы следует ввести с клавиатуры.

Порядок выполнения работы

1. Изучить:

- способы описания структур данных;
- способы ввода и вывода значений элементов структур данных;
- возможности языка Си/Си++ по обработке файлов.

2. Разработать алгоритм решения задачи в соответствии с заданием.

3. Составить программу решения задачи. Программа должна включать в себя три составляющие: объявление структуры данных, функцию обработки данных и бесконечный цикл, в котором осуществляется основная работа программы.

4. Подготовить тестовый файл для проверки правильности функционирования программы.

5. Набрать программу в редакторе системы программирования Microsoft Visual Studio C++.

6. Протестировать разработанную программу. При наличии ошибок внести изменения в программу.

7. Оформить отчет к работе. Отчет должен содержать:

- цель работы;
- задание к работе (с указанием варианта);
- описание алгоритма;
- исходный код программы на языке Си/Си++ (с комментариями);
- содержание тестового файла;
- результаты тестирования (в том числе скриншот работы программы);
- выводы.

Контрольные вопросы и задания

1. Объяснить, что означают следующие термины: файл, запись, метод доступа?

2. Чем отличается файл от массива?

3. В какое место файла можно добавлять новые элементы: в начало, в конец, куда угодно, никуда?

4. Можно ли сравнивать файлы (переменные типа FILE) или присваивать значение одного файла другому?

Глава 7. АЛГОРИТМЫ ПОИСКА И СОРТИРОВКИ ДАННЫХ

7.1. Понятие асимптотической сложности алгоритма

Как известно, одну и ту же задачу можно решить, используя разные алгоритмы. Разные алгоритмы требуют разный объем вычислительных ресурсов, и в первую очередь машинного времени на его выполнения. Полезно проанализировать, сколько вычислительных ресурсов они требуют и выбрать наиболее эффективный. На практике точно оценить время работы программы в секундах или минутах почти невозможно. Традиционно считается, что время работы прямо пропорционально количеству операций. Наибольший интерес вызывает зависимость числа операций конкретного алгоритма от размера входных данных.

Строго говоря, у зависимости времени работы от входных данных есть собственное название – временная сложность алгоритма.

Временная сложность алгоритма – это понятие в информатике и теории алгоритмов, обозначающее зависимость количества элементарных операций, проделываемых алгоритмом для решения экземпляра задачи, от размера входных данных.

Рассмотрение входных данных большого размера и оценка порядка роста времени работы алгоритма, приводит к понятию *асимптотической сложности* алгоритмов.

Анализ сравнения затрат времени алгоритмов, выполняемых решение экземпляра некоторой задачи, при больших объемах входных данных, называется *асимптотическим*. Алгоритм, имеющий наименьшую асимптотическую сложность, является наиболее эффективным.

В асимптотическом анализе, *сложность алгоритма* – это функция, позволяющая определить, как быстро увеличивается время работы алгоритма с увеличением объема данных.

При оценке асимптотической сложности алгоритма обычно используются следующие оценки:

- **O(*f*(*n*))** (О-большое) – верхняя асимптотическая оценка роста временной функции или сложность наиболее неблагоприятного случая, когда алгоритм работает дольше всего;

- **$\Omega(f(n))$** (Омега) – нижняя асимптотическая оценка роста временной функции или сложность в наиболее благоприятном случае, когда алгоритм справляется быстрее всего;
- **$\Theta(f(n))$** (Тета) – сложность алгоритма в среднем.

Под наихудшим случаем понимается время работы алгоритма для любых входных данных размером n. Время работы алгоритма в наихудшем случае – это верхний предел этой величины для любых входных данных.

Располагая этим значением, мы точно знаем, что для выполнения алгоритма не потребуется большее количество времени. Не нужно будет делать каких-то сложных предположений о времени работы и надеяться, что на самом деле эта величина не будет превышена.

Для многих алгоритмов худший случай – это явление крайне редкое, для таких алгоритмов асимптотическая сложность обычно оценивается по среднему значению. Примером такого алгоритма является qsort, который будет рассмотрен в этом разделе ниже.

Под фразой «сложность алгоритма есть $O(f(n))$» подразумевается, что с увеличением объема входных данных n, время работы алгоритма будет возрастать не быстрее, чем некоторая функция $f(n)$.

Чтобы определить сложность алгоритма, не нужно считать все операции, достаточно знать какой сложностью обладает та или иная конструкция алгоритма (оператор или группа операторов). Так, алгоритм, не содержащий циклов и рекурсий, имеет константную сложность $O(1)$. Сложность цикла, выполняющего n итераций, равна $O(n)$. Конструкция их двух вложенных циклов, зависящих от одной и той же переменной n, имеет квадратичную сложность $O(n^2)$.

Вот наиболее часто встречающиеся классы сложности:

- $O(1)$ – константная сложность;
- $O(n)$ – линейная сложность;
- $O(n^a)$ – полиномиальная сложность;
- $O(log(n))$ – логарифмическая сложность;
- $O(n \cdot log(n))$ – квазилинейная сложность;
- $O(2^n)$ – экспоненциальная сложность;
- $O(n!)$ – факториальная сложность.

Алгоритмы поиска и сортировки широко используются в программировании для решения различных задач, где требуется обработка некоторого множества данных с целью выявления подмножества данных, соответствующего критериям поиска (поиск) или упорядочение элементов в подмножестве данных по какому-либо критерию

(сортировка). Существует большое количество различных алгоритмов поиска и сортировки, которые обладают разной эффективностью. Далее будут рассмотрены некоторые из них и приведены оценки их сложности.

7.2. Алгоритмы поиска данных

Простой алгоритм поиска, brute-force

Рассмотрим этот алгоритм на примере следующей задачи: имеется одномерный массив размером N заполненный случайными целыми числами, необходимо найти в этом массиве элемент, имеющий заданное значение (value) и определить его индекс (index). Если элемент не будет найден, тогда присвоить переменной index значение «-1».

Решение этой задачи возможно методом brute-force (метод «грубой силы» или метод перебора).

Суть метода brute-force состоит в том, чтобы последовательно сравнить все элементы массива с заданной величиной value и в случае нахождения соответствия, присвоить переменной index индекс текущей ячейки и прервать цикл. В таблице 7.1 приведен программный код реализации алгоритма поиска brute-force.

Таблица 7.1. Реализация алгоритма поиска brute-force

Код	Комментарии
`#define N 7` …	// размер массива
`int i, index, value = 3;` `int array[N] = {1,7,9,3,2,1,0};`	// объявление // переменных

`index = -1;`
`for (i=0; i<N; i++)`
`{`
`  if (array[i] == value)`
`  {`
`    index = i;`
`    break;`
`  }`
`}`
	// поиск

`printf("Index = %d\n", index);`	// вывод решения

Оценим асимптотическую сложность этого алгоритма.

В худшем случае, сложность алгоритма в данном случае определяется циклом **for** и равняется $O(N)$.

Можно попытаться определить и среднюю сложность, если предположить, что вероятности встретить нужное значение, как в начале массива, так и в конце массива равны, можно сделать вывод о том, что в среднем необходимо выполнить $N/2$ итераций, соответственно средняя сложность алгоритма будет равняться $\Theta(N/2)$. Однако, как следует из определения, асимптотическая сложность обычно округляется «до константы», соответственно, ошибкой не будет, если мы запишем ее, как $\Theta(N)$.

Двоичный поиск, binary-search

Выше мы рассмотрели задачу поиска значения в массиве, заполненном случайными числами. Частным случаем массива, является массив упорядоченный (отсортированный) по возрастанию. В реальной жизни редко встречаются неупорядоченные массивы с множеством элементов: телефонный справочник отсортирован по алфавиты, гайки в строительном магазине отсортированы по диаметру резьбы и т.д.

При работе с отсортированными массивами поиск можно выполнять гораздо быстрее, нежели с помощью алгоритма brute-force. Для этой цели обычно используется алгоритм binary-search (бинарный поиск или поиск «делением пополам») – классический алгоритм, использующий для поиска в отсортированном массиве.

Алгоритм состоит в следующем:

- определение значения элемента в середине структуры данных. Полученное значение сравнивается с искомым значением;
- если искомое значение меньше значения середины то поиск осуществляется в первой половине элементов, иначе – во второй;
- поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с искомым значением;
- процесс продолжается до тех пор, пока не будет найден элемент, совпадающий с искомым значением, или интервал для поиска не станет пустым.

Реализация алгоритма на языке Си/Си++ приведена в таблице 7.2.

Таблица 7.2. Реализация алгоритма поиска binary-search

Код	Комментарии
#define N 7 …	// размер массива
int index, value = 3; int middle, first, last; int array[N] = {0,1,1,2,3,7,9};	// объявление // переменных

index = -1;
first = 0;
last = N;
	// подготовка к
	// бинарному поиску

while (first < last)
{
  middle = first + (last - first) / 2;
  if (value <= array[middle])
    last = middle;
  else
    first = middle + 1;
}
if (last < N && array[last] == value)
  index = last;
	// бинарный поиск

printf("Index = %d\n ", index);	// вывод решения

Асимптотическая сложность алгоритма binary-search, составляет $\Theta(\log_2 N) = O(\log_2 N)$. Таким образом, для нахождения элемента в массиве, имеющем 1млн элементов, потребуется всего 20 итераций!

Следует отметить, что оба рассмотренных нами алгоритма поиска, находят лишь один элемент со значением «value». Если вам требуется найти все элементы со значением «value», вам придется самостоятельно доработать алгоритмы.

Понятие хеширования, нахождение хеш-суммы, hash-sum

При написании алгоритмов частот возникает ситуация, когда программисту приходится оперировать данными достаточно большого объема (например, текстом книг).

Рассмотрим задачу: необходимо сравнить два текста (два массива) и определить, не совпадают ли они?

Эту задачу можно решить алгоритмом brute-force – сравнивая последовательно все элементы массива от начала и до конца (таблица 7.3).

Таблица 7.3. Решение задачи сравнения текста методом brute-force

Код	Комментарии
#define M 7	// размер массива
bool is_equal; int i; int array_1[M] = {1,7,9,3,2,1,0}; int array_2[M] = {1,7,9,8,9,1,0};	// объявление // переменных

is_equal = true;
for (i=0; i<M; i++)
{
  if (array_1[i] != array_2[i])
  {
    is_equal = false;
    break;
  }
}
	// поиск несовпадений

if (is_equal == true)
  printf("Arrays are equal\n");
else
  printf("Arrays are different\n");
	// вывод решения

На выполнение подобного алгоритма потребуется время M (т.е. асимптотическая сложность алгоритма – $O(M)$), пропорциональное длине массивов. Если в ходе работы программы требуется выполнить множество подобных операций, для ускорения алгоритма можно воспользоваться хешированием.

Хеширование – преобразование массива входных данных произвольной длины в выходную битовую строку фиксированной длины, выполняемое определенным алгоритмом.

Если выполнить предварительное хеширование массивов, то операция сравнения двух массивов будет выполняться за время O(1). В таблице 7.4 приведен программный код сравнения текста с помощью Хэш-суммы и наиболее простой алгоритм расчета Хеш-суммы – сложением всех элементов массива.

Таблица 7.4. Решение задачи сравнения текста с помощью Хэш-суммы

Код	Комментарии
#include "stdafx.h" #include <stdio.h> #include <string.h>	// подключение библиотек

int hash(char* str)
{
  int i, ans = 0;
  for (i = 0; i < strlen(str); i++)
  {
    ans += str[i];
  }
  return ans;
}
	// функция вычисление
	// Хэш-суммы

int main(void)
{
  char text_1[10] = "hello";
  char text_2[10] = "world";
 	
  int text_1_hash = hash(text_1);
  int text_2_hash = hash(text_2);
 	
  if (text_1_hash == text_2_hash)
    printf("Strings are equal\n");
  else
    printf("Strings are different\n");
 	
  getchar();
  fflush(stdin);
  return 0;
}
	// основная функция
	
	// пример текста, который
	// требуется сравнить
	
	// вычисление Хэш-сумм
	
	// сравнение Хэш-сумм

Недостатком этого простого алгоритма нахождения хеш-суммы (или контрольной суммы) является высокая вероятность коллизий – случая, когда одной и той же хеш-сумме соответствуют разные массивы.

Подобного недостатка лишены алгоритмы CRC32, MD5 и ряд других (эти алгоритмы следует изучить самостоятельно, обратившись к литературе). Однако следует всегда помнить, что коллизий при хешировании на 100% избежать невозможно, т.к. мощность множества

комбинаций исходных текстов которые необходимо хэшировать всегда больше мощности множества комбинаций хеш-сумм.

Следует помнить, что расчет хеш-суммы является достаточно ресурсозатратной операцией, по этой причине рекомендуется хранить эти суммы в отдельном файле, чтобы каждый раз при запуске программы не приходилось выполнять лишнюю работу.

Хеширование, как метод ускорения поиска

При оценке сложности алгоритмов поиска brute-force и binary-search мы предполагали, что сравнение двух элементов массива выполняется за время O(1). Однако когда мы выполняем поиск, например в телефонном справочнике, время сравнения составляет $O(M)$, где M – длина строки в справочнике.

Принимая к сведению вышесказанное, можно определить время поиска в телефонном справочнике: для алгоритма brute-force оно составляет $O(M \cdot N)$, а для алгоритма binary-serach $O(M \cdot \log_2 N)$. Значительно ускорить процедуру поиска информации в справочнике, можно за счет хеширования, асимптотическая сложность алгоритма при использовании хеширования снизится в M-раз.

7.3. Алгоритмы сортировки данных

Очень часто на практике встречаются задачи, в которых требуется переставить элементы неупорядоченного массива, после чего они становятся упорядоченными по возрастанию или убыванию. В параграфе 7.2 отмечалось, что поиск данных в отсортированном массиве выполняется значительно быстрее, нежели в неотсортированном. Подобная ситуация наблюдается и в реальной жизни, найти какую-то вещь среди беспорядка обычно бывает весьма затруднительно.

Сортировка элементов одномерного массива – это такая перестановка его элементов, в результате которой они оказываются упорядоченными определенным образом.

Имеется большой выбор алгоритмов сортировки, в которых применяются самые разные технологии. Фактически в алгоритмах сортировки используются многие важные методы, применяемые при разработке различных классов алгоритмов.

Рассмотрим задачу сортировки в простейшей постановке: дан числовой массив $x_1, \ldots, x_n$, элементы которого попарно различны; тре-

буется переставить элементы массива так, чтобы после перестановки они были упорядочены в порядке возрастания: $x_1 < ... < x_n$.

Существует несколько способов решения задачи сортировки. Каждый из них требует выполнения различного количества операций а, следовательно, времени расчета и памяти ЭВМ, необходимой для их реализации.

Рассмотрим следующие виды сортировок:

- вставкой;
- обменом ("пузырьковая сортировка");
- быстрая сортировка;
- сортировка подсчетом.

Алгоритм сортировки простыми вставками, insertion-sort

Одним из наиболее простых алгоритмов сортировки является алгоритм сортировки простыми вставками (insertion-sort). Сортировка вставками – алгоритм сортировки, в котором элементы массива просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

Иными словами на первом шаге алгоритма, предполагается, что массив, включающий в себя первый символ, является отсортированным, а затем в него по одному вставляются новые элементы из неотсортированной части массива. Это происходит до тех пор, пока весь массив не будет отсортирован.

Таким образом, будет иметь место (N-1) проходов (где N – размерность массива), каждый из которых будет включать четыре действия:

- взятие очередного элемента из неотсортированной части и сохранение в дополнительной переменной;
- поиск позиции в отсортированной части массива для вставки элемента, взятого предыдущим действием, который бы не нарушил порядок;
- сдвиг элементов массива для вставки элемента массива в отсортированную часть;
- вставка элемента в найденную позицию.

В таблице 7.5 приведен пример сортировки методом insertion-sort.

Таблица 7.5. Пример сортировки методом insertion-sort

Итерация 1	1	7	9	12	3
Итерация 2	1	→	7 →	9 →	12
Итерация 3	1	3	7	9	12

В примере серым обозначена уже отсортированная часть массива, а белым – неотсортированная. На первой итерации производится перенос переменной имеющей значение 3 в буфер, на второй итерации – сдвиг элементов массива право, начиная с элемента, на место которого должен перейти элемент 3. На третьей итерации производится вставка элемента из буфера обмена на свою позицию в отсортированном массиве.

В таблице 7.6 приведена программа на Си/Си++ реализующая указанный алгоритм.

Таблица 7.6. Реализация алгоритма сортировки методом insertion-sort

Код	Комментарии
`#define N 5`	// размер массива
`int i, j, key;` `int array[N] = {1,7,9,12,3};`	// объявление // переменных

`for (j=1; j<N; j++)`
`{`
`  key = array[j];`
`  i = j-1;`
`  while(i>=0 && array [i] > key)`
`  {`
`    array [i+1] = array [i];`
`    i--;`
`  }`
`  array [i+1] = key;`
`}`
	// сортировка
	// вставками

У сортировки вставками имеются два ключевых недостатка:

- высокая асимптотическая сложность алгоритма: $\Theta(n^2)$;
- не смотря на простоту алгоритма, его воспроизведение по памяти обычно требует достаточно много времени.

Алгоритм сортировки обменом, bubble-sort

С точки зрения реализации наиболее простым алгоритмом сортировки является т.н. алгоритм пузырьковой сортировки (bubble-sort).

Данный алгоритм считается учебным и практически не применяется вне учебной литературы, т.к. *его асимптотическая сложность* – $\Theta(n^2)$, вместо него на практике применяются более эффек-

тивные алгоритмы сортировки. В то же время метод сортировки обменами лежит в основе некоторых более совершенных алгоритмов, таких как пирамидальная сортировка и быстрая сортировка, о которых будет сказано ниже.

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.

Метод «пузырька» предполагает следующую модель преобразования массива данных.

Из *N* чисел массива всплывает (как в бокале с шампанским) «пузырек» – число, оказывающееся больше в случае сортировки по возрастанию, чем (*N*-1) чисел. Затем всплывает «пузырек» - число, оказывающееся больше чем (*N*-*2*) оставшихся, но меньше, чем первый всплывший «пузырек» и т.д.

В таблице 7.7 приведен пример сортировки этим алгоритмов.

Таблица 7.7. Пример сортировки методом bubble-sort

Итерация 1	1	7	9	12 ↔	3
	1	7	9	3	12
Итерация 2	1	7	9 ↔	3	12
	1	7	3	9	12
Итерация 3	1	7 ↔	3	9	12
	1	3	7	9	12

В таблице 7.8 приведена программа на Си/Си++ реализующая указанный алгоритм.

Таблица 7.8. Реализация алгоритма сортировки методом bubble-sort

Код	Комментарии
#define N 5	// размер массива
int i, j, key; int array[N] = {1,7,9,12,3};	// объявление // переменных

for (i=0; i<N; i++)
{
for (j=0; j<N-1; j++)
if (array [j]> array [j+1])
{
key = array [j];
array [j] = array [j+1];
array [j+1] = key;
}
}
	// пузырьковая
	// сортировка

Приведенная выше реализация является наименее эффективной, но наиболее простой реализацией данного алгоритма. Если требуется быстро написать собственную реализацию алгоритма сортировки, то bubble-sort – лучший выбор.

Изменив знак в строке сравнения, можно изменить порядок сортировки (возможна как сортировка по возрастанию, так и по убыванию).

Быстрая сортировка, quick-sort и ее конкуренты

Наиболее быстрые алгоритмы сортировки имеют асимптотическую сложность $O(n \cdot log_2 n)$. Простейшим представителем этого семейства алгоритмов является т.н. алгоритм быстрой сортировки (quick-sort). Данный алгоритм основан на принципе «разделяй и властвуй», отсюда, наиболее простой его реализацией является рекурсивная.

Принцип реализации алгоритма состоит в следующем:

- *Разделение*. Массив разбивается на два (возможно, пустых) подмассива и элемент q, при этом каждый из элементов первого массива меньше или равен q, а каждый из элементов второго массива больше или равен q. Элемент q определяется в ходе процедуры разбиения.
- *Властвование*. Подмассивы, полученные на предыдущем этапе, сортируются с помощью рекурсивного вызова процедуры быстрой сортировки.
- *Комбинирование*. Поскольку подмассивы сортируются на месте, для их объединения не требуются никакие действия: весь массив оказывается отсортированным.

Программа, реализующая быструю сортировку, представлена в таблице 7.9.

Таблица 7.9. Реализация алгоритма сортировки методом quick-sort

Код	Комментарии
#include "stdafx.h" #include <stdio.h>	// подключение // библиотек
#define N 5	// размер // массива
int array[N] = {1,7,9,12,3};	// объявление // переменных

Код	Комментарии

int partition(int *array, int p, int r)
{
int x = array[r], key;
int i = p - 1, j;
 	
for (j = p; j <= r - 1; j++)
{
if (array[j] <= x)
{
i++;
key = array[i];
array[i] = array[j];
array[j] = key;
}
}
key = array[i + 1];
array[i + 1] = array[r];
array[r] = key;
return i + 1;
}
	// функция
	// разделения

void qsort(int *array, int p, int r)
{
int q;
 	
if (p<r)
{
q = partition(array, p, r);
qsort(array, p, q-1);
qsort(array, q+1, r);
}
}
	// функция
	// сортировки

int main(void)
{
int i;
for (i = 0; i<N; i++) printf("%d ", array[i]);
qsort(array, 0, N - 1);
printf("\n");
for (i = 0; i<N; i++) printf("%d ", array[i]);
fflush(stdin);
getchar();
return 0;
}
	// основная
	// функция

Алгоритм quick_sort является достаточно простым алгоритмом, он также хорош с точки зрения потребляемой памяти, однако асимптотическая сложность данного алгоритма относительно низкая лишь в среднем случае, тогда как в наихудшем случае она составляет $\Theta(n^2)$. По этой причине, прежде чем выполнять сортировку рекомендуется провести рандомизацию массива (переставить все элементы массива на произвольные позиции).

Альтернатив у алгоритма быстрой сортировки существует несколько, однако все они отличаются сложность реализации и их следует рассмотреть самостоятельно (они описаны в [3]). К таким алгоритмам относится, например пирамидальная сортировка, имеющая асимптотическую сложность $O(n \cdot log_2 n) = \Theta(n \cdot log_2 n)$. Главным недостатком этого алгоритма является значительный объем ОЗУ, который будет занимать программа во время работы сортировки.

Сортировка за линейное время, counting-sort

Сортировку в некоторых случаях можно выполнить и за линейное время, т.е. асимптотическая сложность алгоритма будет $\Theta(n)$, однако для достижения таких показателей сортируемый массив должен обладать несколькими особыми свойствами:

- элементы массива должны быть целочисленными
- разница между значением максимального и минимального элементов массива (m) намного меньше размера массива (n).

Сортировка за линейное время называется сортировкой подсчетом (counting-sort). Асимптотическая сложность указанного алгоритма, строго говоря $\Theta(m+n)$, однако когда $m<<n$, считывается, что сложность алгоритма $\Theta(n)$.

Алгоритм сортировки подсчетом состоит из трех частей:

- обнуление массива-счетчика размером m;
- подсчет в массиве-счетчике того, какие числа и сколько раз встречаются в основном массиве;
- составление ответа на основании данных массива-счетчика.

В таблице 7.10 приводится реализация указанного алгоритма.

Таблица 7.10. Реализация алгоритма сортировки методом counting-sort

Код	Комментарии
#define N 5 #define M 13	// размер массивов
int i, j, key; int A[N] = {1,7,9,12,3}; int counter[M];	// объявление // переменных
for(i=0; i<M; i++) counter[i] = 0;	// обнуление счетчика
for (i=0; i<N;i++) counter[A[i]]++;	// подсчет элементов

for (i=0; i<M; i++)
  for (j=0; j<counter[i]; j++)
    printf("%d ", i);
	// вывод решения

Данный пример не является оптимальным для алгоритма counting-sort, для выполнения алгоритма потребовалось $m+n+(m+n)$ операций, итого 36 операций, что даже больше, чем у пузырькового метода. Однако если бы мы рассматривали случай, когда $n = 1000$, а $m = 10$, то на выполнение алгоритма counting-search потребуется всего 2200 операций, что меньше чем у quick-sort (10 тыс. операций) и меньше, чем у bubble-search (1 млн операций).

Мы рассмотрели некоторые наиболее популярные алгоритмы по работе с массивами. В Си/Си++ можно работать и с другими сложными типами данных – со стеками, деками, очередями, деревьями, графами и многими другими сложными типами данных. Алгоритмы работы с этими, а также многими другими типами рассматриваются в [3].

Вопросы для самопроверки

1. Что такое асимптотическая сложность алгоритма?
2. Перечислите особенности алгоритма поиска brute-force.
3. Перечислите особенности алгоритма поиска binary-search.
4. Что такое hash-сумма? Приведите пример вычисления hash-суммы.
5. За счет чего хеширование ускоряет выполнение алгоритмов?
6. Перечислите особенности алгоритма поиска insertion-search.
7. Перечислите особенности алгоритм поиска bubble-search.

8. Перечислите особенности алгоритм поиска quick-search.
9. Какие алгоритмы поиска за время O(n·log2n) вы знаете?
10. В каком случае алгоритм поиска counting-search будет медленнее алгоритма bubble-search?

Практические задания

1. Необходимо написать программу на языке Си, которая генерирует массив из 25 элементов, а затем сортировку его по четности, т.е. в левой стороне массива должны быть нечетные числа, а в правой стороне – четные.

2. Необходимо написать программу на языке Си, которая подготавливает массив, состоящий из случайных чисел для последующего поиска в нем методом binary-search.

Примечание: необходимо составить как алгоритм предварительной обработки, так и алгоритм поиска (binary-search).

3. Необходимо написать программу на языке Си, которая включает в себя массив из строк, который представляет собой список фамилий известных писателей. Необходимо отсортировать этот список в лексикографическом порядке, т.е. от «А» до «Я».

Примечание: если все фамилии будут начинаться с разных букв, алгоритм можно значительно упростить.

4. Необходимо написать программу на языке Си, которая включает в себя массив из строк, представляющий собой список фамилий известных писателей. Для каждой фамилии требуется найти хеш-сумму (методом суммы, из параграфа 7.2), а затем определить, какой строке принадлежит хеш-сумма -228 (например).

5. Необходимо написать программу на языке Си, для решения нижеизложенной задачи. Имеется хеш-функция из параграфа 7.2 (сумма). Имеется строка «Достоевский Федор Михайлович». Необходимо написать программу, которая вычисляет хеш-сумму заданной строки, а затем пытается искусственно подобрать другую строку имеющую туже хеш-сумму.

ЗАКЛЮЧЕНИЕ

Используемый в пособии учебный материал, в том числе и задачи для самостоятельного решения и варианты индивидуальных заданий, подобраны с учетом того, что дисциплины «Программирование и основы алгоритмизации» и «Алгоритмизация и программирование» изучаются на первом курсе и, как правило, на этом этапе обучения уровень подготовки студентов в данной области очень разный.

Полученные знания будут использованы студентами в специальных дисциплинах при разработке алгоритмов и программ, реализующих процессы обработки цифровой информации.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Абрамян М.Э. 1000 задач по программированию. Часть I. Скалярные типы данных, управляющие операторы, процедуры и функции. – Ростов-на-Дону: РГУ, 2004. – 43 с.
2. Леонов Ю.Г., Глазунова Л.В. и др. Сборник задач по программированию. – Одесса: ОНАС им. А.С. Попова, 2011. – 212 с.
3. Кормен Т., Лейзерсон Ч. и др. Алгоритмы, построение и анализ. Издание 3-е. – М.: Вильямс, 2013. – 1924с.
4. Кнут Д.Э. Искусство программирования. Т.4, А. – М.: Вильямс, 2013. – 960с.
5. Нейбауэр А. Моя первая программа на С и С++. – Спб: Питер, 2002. – 267с.
6. Шилдт Г. Полный справочник по С++, 4-е издание. – М.: Вильямс, 2006. – 800с.
7. Кунтин Н. С/С++ в задачах и примерах. – Спб.: БХВ-Петербург, 2009. – 368 с.

ПРИЛОЖЕНИЕ

ОСНОВЫ ПРАКТИЧЕСКОЙ РАБОТЫ В ИНТЕГРИРОВАННОЙ СРЕДЕ РАЗРАБОТКИ MICROSOFT VISUAL STUDIO C++

Интегрированная среда разработки Microsoft Visual Studio C++ состоит из компилятора, редактора исходных кодов, отладчика и набора вспомогательного программного обеспечения.

Ниже приводятся краткие инструкции по созданию приложения и первому запуску приложения. Более подробные сведения содержатся, например в [5 – 7].

1. Создание проекта в Microsoft Visual Studio C++

Для создания приложения, в Visual Studio C++ нужно выполнить следующую последовательность действий:

1. Запустить MS Visual Studio и нажать на кнопку **Создать проект** (рисунок П1).

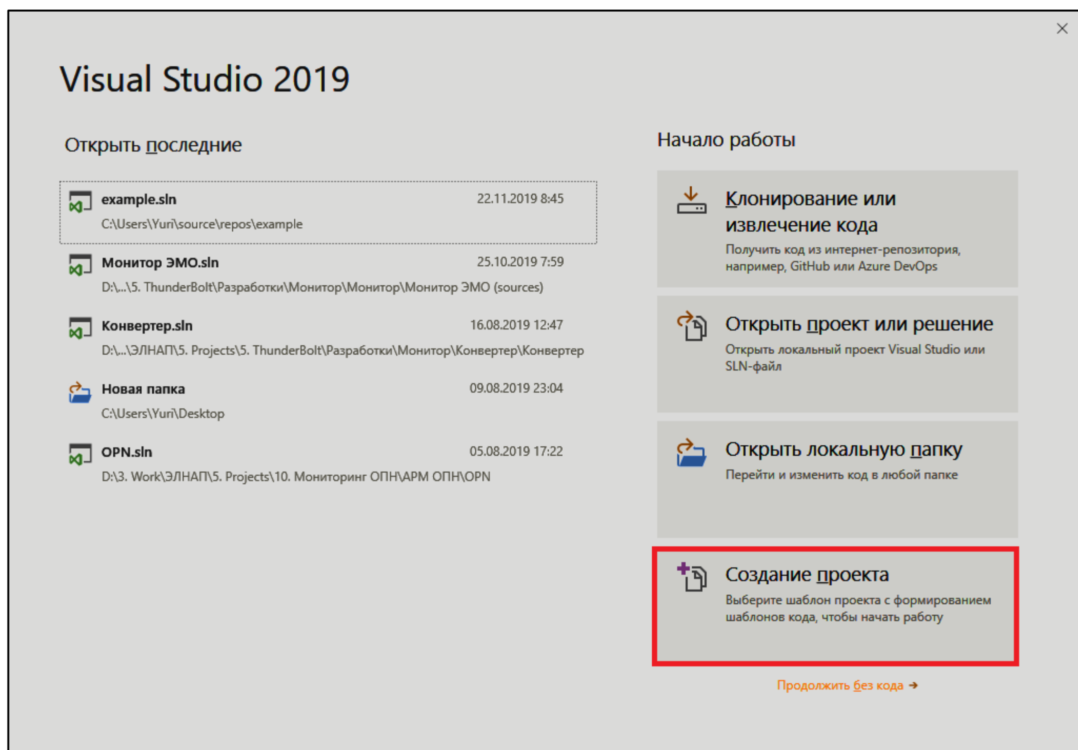

Рисунок П1 – Создание проекта в MS Visual Studio

2. Указать тип проекта – **Консольное приложение** (для этого необходимо предварительно ввести в строке поиска «консольное приложение с++») и нажать кнопку **Далее** (рисунок П2).

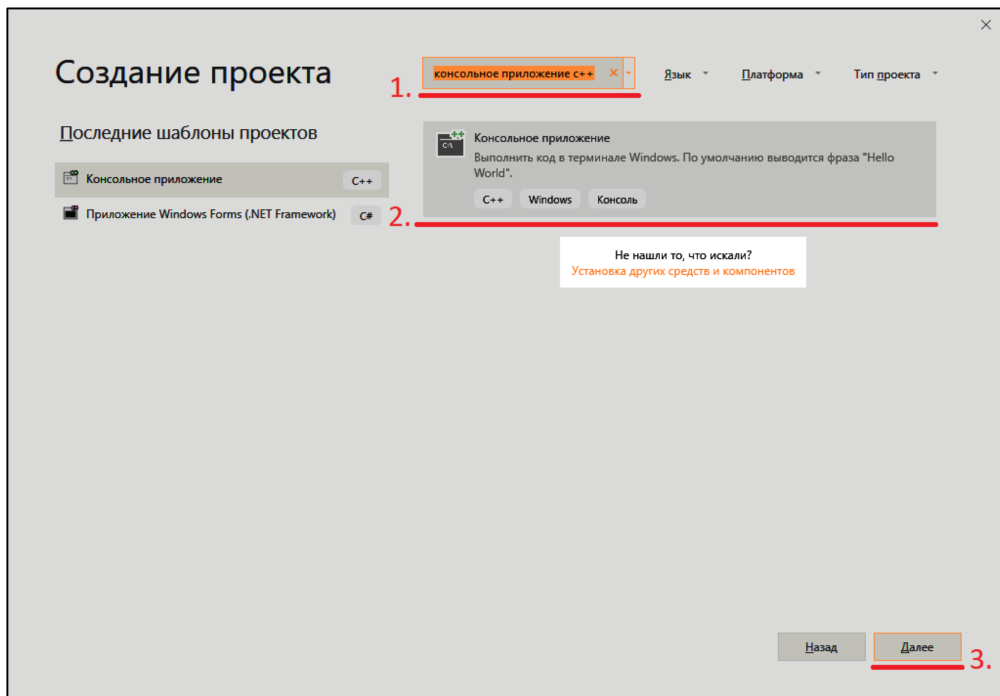

Рисунок П2 – Выбор типа проекта MS Visual Studio

3. Указать **Имя** и **Расположение** проекта, после чего нажать кнопку **Создать** (рисунок П3).

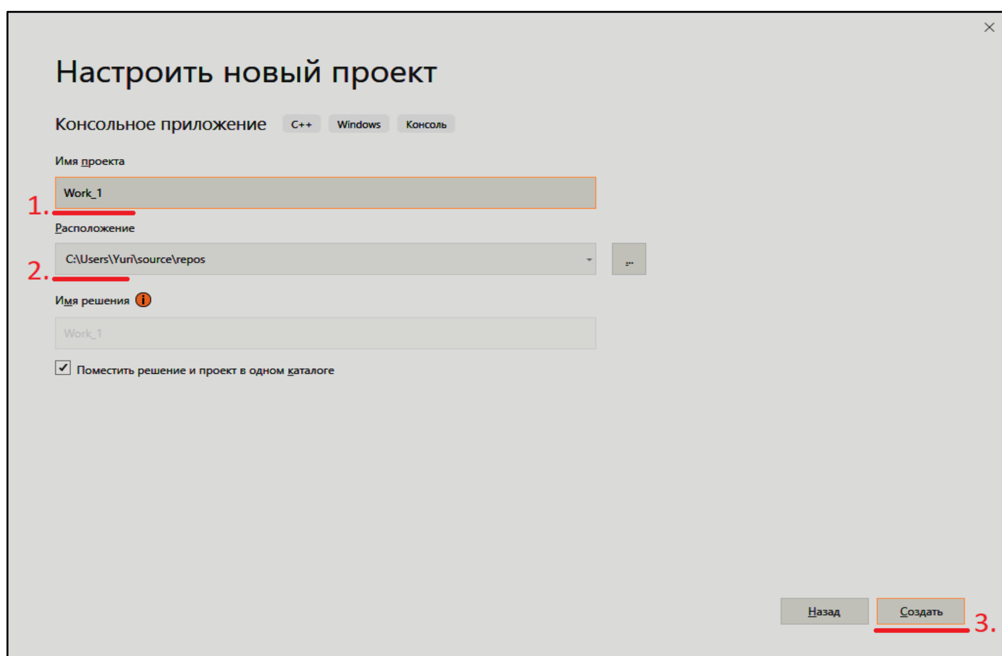

Рисунок П3 – Указание названия и расположения проекта

4. Ввести код приложения в открывшийся файл ***.cpp**.
5. Сохранить файл (комбинация клавиш **Ctrl** + **S**).
6. Для запуска, нажать на кнопку **Локальный отладчик Windows** (рисунок П4).

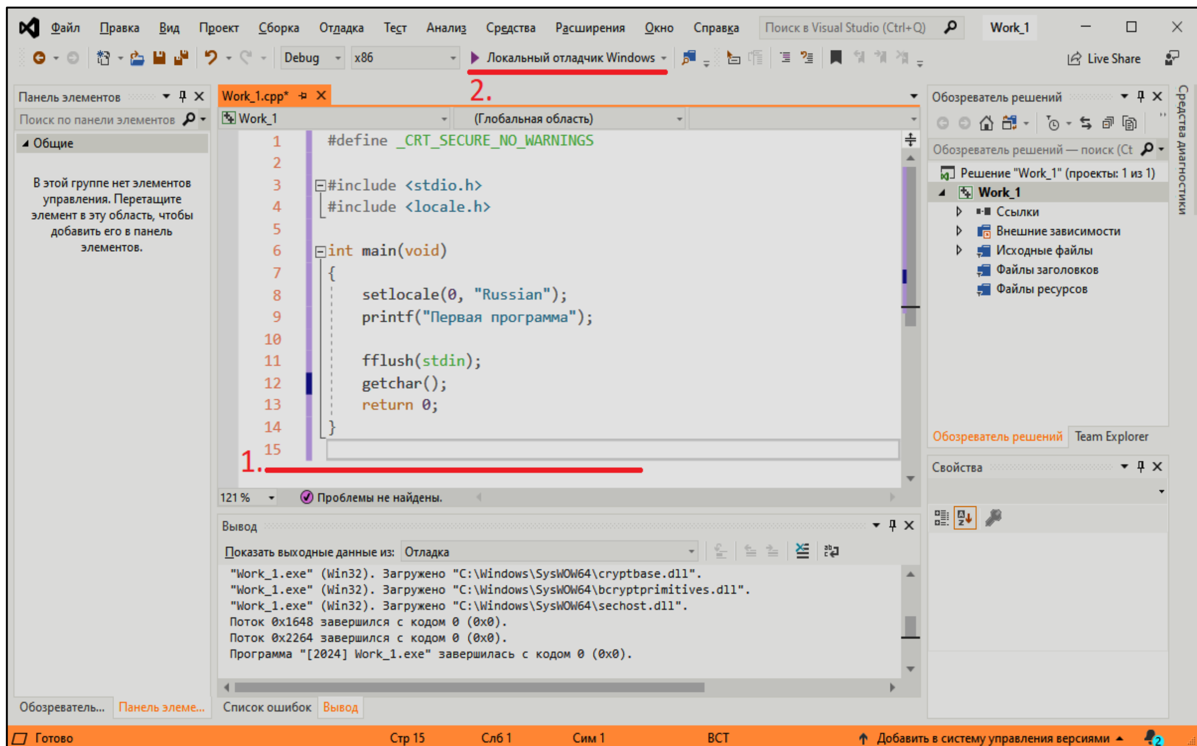

Рисунок П4 – Редактирование и запуск программы

2. Первая программа в среде Microsoft Visual Studio C++

В таблице П1 приведен пример, который можно использовать в качестве первой программы на языке Си/Си++. В окно редактора среды Microsoft Visual Studio C++ следует вводить только код из указанной таблицы.

Таблица П1. Пример программы на языке Си/Си++

Код	Комментарии
//#include "stdafx.h" #include <stdio.h> #include <locale.h>	// используется в некоторых версия Visual Studio // подключение библиотек
int main(void) {	// основная функция

Код	Комментарии
setlocale(0, "Russian");	// вкл. русского языка
printf("Первая програм-ма");	// вывод текста
fflush(stdin); getchar();	// пауза // аналог getchar, для новых версий Visual Studio
return 0; }	// завершение программы

3. Разрешение проблемы консольного ввода

При работе в современной среде разработки, например в Visual Studio 2015, для нормальной работы функции **scanf()** может потребоваться произвести дополнительные действия: отредактировать файл **stdafx.cpp** согласно с образцом из таблицы П2.

Таблица П2. Файла конфигурации stdafx.cpp

Код	Комментарии
#define _CRT_SECURE_NO_WARNINGS	// отключение системы // предупреждений CRT

В среде Visual Studio 2017 и 2019 директиву из таблицы 10 следует вводить непосредственно в главный ***.cpp** файл проекта (в тот же файл, в котором содержится функция **main**) причем в самом начале файла (строго в первой строке).

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ 3

Глава 1. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ
И ОСНОВЫ АЛГОРИТМИЗАЦИИ 4
1.1. Классификация языков программирования 4
1.2. Понятие алгоритма 7
1.3. Базовая структура программы на языке Си/Си++ 10
Вопросы для самопроверки 15
Практические задания 15

Глава 2. ОБРАБОТКА ДАННЫХ. МАТЕМАТИЧЕСКИЕ
И ЛОГИЧЕСКИЕ ОПЕРАЦИИ 16
2.1. Типы данных 16
2.2. Простые типы данных 17
2.3. Ввод и вывод данных в консоль 22
2.4. Простые операции обработки данных, математические
и логические операции 28
Вопросы для самопроверки 33
Практические задания 34
Практическая работа № 1. Программирование алгоритмов
линейной структуры 35

Глава 3. РЕАЛИЗАЦИЯ ВЕТВЯЩИХСЯ И ЦИКЛИЧЕСКИХ
АЛГОРИТМОВ 39
3.1. Реализация ветвящихся алгоритмов 39
3.2. Реализация циклических алгоритмов 46
Вопросы для самопроверки 53
Практические задания 53
Практическая работа № 2. Программирование алгоритмов
разветвляющейся и циклической структуры 55
3.3. Рекуррентные вычисления 59
Вопросы для самопроверки 61
Практическая работа № 3. Программирование алгоритмов
с вложенным циклом 62

Глава 4. СЛОЖНЫЕ ТИПЫ ДАННЫХ, СТРУКТУРЫ 67
4.1. Одномерные массивы, определение и объявление 67
4.2. Алгоритмы типовых действий с одномерными массивами 70
Вопросы для самопроверки 73
Практические задания 74
Практическая работа № 4. Обработка одномерных массивов.......... 74
4.3. Многомерные массивы 77
4.4. Алгоритмы типовых действий с двумерными массивами (матрицами).. 79
Вопросы для самопроверки 88
Практические задания 88
Практическая работа № 5. Обработка матриц................................. 88
4.5. Строка как особый вид массива 91
Вопросы для самопроверки 95
Практические задания 95
Практическая работа № 6. Обработка символьных и строковых данных 95
4.6. Структуры данных 98
Вопросы для самопроверки 105
Практические задания 105

Глава 5. ПОДПРОГРАММЫ, ФУНКЦИИ, МОДУЛИ...................... 106
5.1. Понятие подпрограммы, функции и процедуры 106
5.2. Функции 107
5.3. Рекурсивные функции 112
5.4. Внешние модули, библиотеки 115
Вопросы для самопроверки 115
Практические задания 116
Практическая работа № 7. Программирование с использованием функций пользователя 116

Глава 6. РАБОТА С УКАЗАТЕЛЯМИ И ФАЙЛАМИ 121
6.1. Работа с указателями 121
Вопросы для самопроверки 127
Практические задания 127
6.2. Работа с файлами 127
Вопросы для самопроверки 135
Практические задания 135
Практическая работа № 8. Работа со структурами данных
и файлами 135

Глава 7. АЛГОРИТМЫ ПОИСКА И СОРТИРОВКИ ДАННЫХ 140
7.1. Понятие асимптотической сложности алгоритма 140
7.2. Алгоритмы поиска данных 142
7.3. Алгоритмы сортировки данных 147
Вопросы для самопроверки 154
Практические задания 155

ЗАКЛЮЧЕНИЕ 156

БИБЛИОГРАФИЧЕСКИЙ СПИСОК 156

ПРИЛОЖЕНИЕ. ОСНОВЫ ПРАКТИЧЕСКОЙ РАБОТЫ
В ИНТЕГРИРОВАННОЙ СРЕДЕ РАЗРАБОТКИ
MICROSOFT VISUAL STUDIO C++ 157
1. Создание проекта в Microsoft Visual Studio C++ 157
2. Первая программа в среде Microsoft Visual Studio C++ 159
3. Разрешение проблемы консольного ввода 160

Учебное издание

ГРАДУСОВ Александр Борисович
ТИХОНОВ Юрий Васильевич

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ

Учебно-практическое пособие

Издается в авторской редакции

Подписано в печать 26.02.20.
Формат 60x84/16. Усл. печ. л. 9,53. Тираж 50 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.