

Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

Н. В. Пермякова

ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

Методические указания к лабораторным работам
и организации самостоятельной работы
для студентов направления «Бизнес-информатика»
(уровень бакалавриата)

Томск
2022

УДК 004.432.2(811.93)
ББК 32.973.2 (74.202.4)
П26

Рецензент:

Морозова Ю.В., доцент кафедры автоматизации обработки информации ТУСУР, канд. техн. наук

Пермякова, Наталья Викторовна

П26 Информатика и программирование: методические указания к лабораторным работам и организации самостоятельной работы для студентов направления «Бизнес-информатика» (уровень бакалавриата) / Н. В. Пермякова. – Томск : Томск. гос. ун-т. систем упр. и радиоэлектроники, 2022. – 63 с.

Настоящее учебно-методическое пособие по выполнению лабораторных работ и организации самостоятельной работы составлено с учетом требований федерального государственного образовательного стандарта высшего образования (ФГОС ВО).

Учебно-методическое пособие содержит краткое описание теоретического материала, необходимого для выполнения указанных видов занятий. Для каждой лабораторной работы определена цель работы, порядок ее выполнения. Для организации самостоятельной работы перечислены темы контрольных работ и темы курса, предназначенные для самостоятельного изучения, перечислены дополнительные литературные источники.

Одобрено на заседании каф. АОИ протокол № 1 от 20.01.2022

УДК 004.432.2(811.93)
ББК 32.973.2 (74.202.4)

© Пермякова Н.В., 2022
© Томск. гос. ун-т. систем
упр. и радиоэлектроники,
2022

Оглавление

ВВЕДЕНИЕ	4
1 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ	5
1.1 Общие положения	5
1.2 Лабораторная работа «Простая программа на языке Си»	5
1.3 Лабораторная работа «Условные алгоритмы. Проверка ошибок ввода данных.»	8
1.4 Лабораторная работа «Условные алгоритмы. Проверка условий. Геометрия на плоскости»	11
1.5 Лабораторная работа «Итерационные алгоритмы. Вычисление суммы бесконечного ряда»	14
1.6 Лабораторная работа «Обработка массивов. Поиск»	15
1.7 Обработка массивов. Динамические массивы	17
1.8 Лабораторная работа «Обработка матриц. Проектирование пользовательских функций»	23
1.9 Лабораторная работа «Текстовые файлы»	24
1.10 Лабораторная работа «Синтаксис ООП»	25
1.11 Лабораторная работа «Создание класса»	26
1.12 Лабораторная работа «Описание методов»	31
1.13 Лабораторная работа «Конструкторы и деструкторы»	32
1.14 Лабораторная работа «Массивы объектов»	36
1.15 Лабораторная работа «Перегрузка операторов»	40
1.16 Лабораторная работа «Наследование. Создание базового класса и класса-наследника»	45
1.17 Лабораторная работа «Работа с потоковыми классами C++»	49
2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ	52
2.1 Общие положения	52
2.2 Проработка лекционного материала, подготовка к контрольным работам, лабораторным работам и практическим занятиям	52
2.3 Выполнение домашних заданий и контрольных работ	53
2.4 Самостоятельное изучение тем теоретической части курса	53
2.5 Подготовка к зачету с оценкой	57
3 РЕКОМЕНДУЕМЫЕ ИСТОЧНИКИ	58
ПРИЛОЖЕНИЕ А	59

ВВЕДЕНИЕ

В методических указаниях к проведению лабораторных работ и организации самостоятельной работы по дисциплине «Информатика и программирование» собраны материалы для методической поддержки аудиторных занятий и самостоятельной работы студентов.

Целью проведения лабораторных работ и самостоятельной работы является формирование навыков разработки алгоритмов решения задач и навыков объектно-ориентированного анализа решаемых задач. Для достижения заданной цели во время проведения занятий последовательно решаются следующие задачи:

- изучение принципов структурного программирования;
- получение навыков реализации последовательных, условных и циклических алгоритмов на языке программирования Си;
- получение навыков разработки объектно-ориентированных программ;
- изучение средств языка программирования С++ для реализации объектно-ориентированных программ.

По окончании обучения дисциплины «Информатика и программирование» студент должен:

- **знать** принципы структурного и объектно-ориентированного программирования, синтаксис и алфавит языков программирования Си++/Си, основные приемы алгоритмизации;
- **уметь** использовать структурный и объектно-ориентированный подходы к разработке программ; умеет разрабатывать алгоритмы поставленных задач;
- **владеть** навыками разработки алгоритмов и их реализации на языке Си/Си++ для решения задач профессиональной деятельности, в том числе в сфере ИКТ.

Цикл лабораторных работ по дисциплине, выполняемых студентами во время первого семестра, направлен на изучение и закрепление навыков использования конструкций структурного программирования. Лабораторные работы, проводимые в рамках второго семестра, посвящены получению навыков объектно-ориентированного программирования.

Согласно действующему Федеральному Государственному стандарту высшего образования (ФГОС ВО) значительная доля курса отведена на самостоятельную работу студентов. При изучении дисциплины «Информатика и программирование» самостоятельная работа студентов включает несколько видов деятельности – проработка лекционного материала, подготовка к контрольным работам, выполнение домашних заданий, самостоятельное изучение тем, подготовка к зачету с оценкой. В методическом пособии предложены рекомендации по организации перечисленных видов деятельности, включающие порядок выполнения отдельных видов самостоятельной работы, краткие теоретические сведения, перечень литературных источников по темам дисциплины, вынесенным на самостоятельное изучение.

1 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

1.1 Общие положения

Целью проведения лабораторных работ в двух первых семестрах является формирование и развитие навыков структурного и объектно-ориентированного программирования.

Основной формой проведения лабораторных работ является разработка алгоритма решения индивидуальной задачи и его программная реализация на языках C\C++. Процесс программной реализации включает в себя написание программы, отладку программы и тестирование программы.

К основным способам контроля формирования компетенций при выполнении лабораторных работ относятся: индивидуальная защита выполненной работы, организация опроса студентов по теоретическому материалу дисциплины, практическое применение которого осуществляется в ходе выполнения лабораторной работы.

Для получения максимальной оценки за лабораторную работу необходимо выполнить и защитить работу во время, отведенное для ее выполнения, согласно расписанию занятий. Допускается досрочное выполнение лабораторной работы по предварительной договоренности с преподавателем.

Выполнение всех лабораторных работ, предусмотренных рабочей программой дисциплины, является условием допуска к итоговому контролю изучения дисциплины – зачету с оценкой.

1.2 Лабораторная работа «Простая программа на языке Си»

Цель работы: ознакомиться с интегрированной средой Dev-C++, изучить основные типы данных языка Си, функции ввода и вывода информации, получить навыки написания программ на языке Си.

Форма проведения: выполнение индивидуального задания.

Подготовка к выполнению лабораторной работы: для выполнения лабораторной работы необходимо изучить теоретический материал, изложенный в [1]. Описание процесса создания проекта в IDE Dev-C++ рассматривается в главе 2 пособия (стр. 33 – 35). Описание структуры простой программы (стр. 81 – 82) и универсальных функций ввода-вывода информации (стр. 77 – 81) в главе 5 пособия. Для реализации индивидуального задания необходимо ознакомиться с операторами языка Си (стр. 52 – 55) и основными типами данных (стр. 59 – 60).

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Создать проект в Dev-C++.
3. Написать программу на языке Си, выполняемые функции которой могут быть описаны следующей последовательностью шагов:
 - 3.1) описать входные и выходные данные;
 - 3.2) ввести данные с клавиатуры;
 - 3.3) вычислить значение функции;
 - 3.4) вывести полученное значение на экран;
 - 3.5) вывести личные данные.
4. Выполнить компиляцию проекта.
5. Выполнить тестирование проекта.
6. Защитить работу.

Контрольные вопросы

1. Какое имя носит исполняемая функция Си?
2. Дайте определение понятия «переменная».
3. Дайте определение понятия «идентификатор».
4. Сколько переменных требуется описать в программе, если необходимо решить следующую задачу – «С клавиатуры вводятся три числа, необходимо вывести на экран значение минимального из этих трех чисел»?
5. Какая функция используется в Си для ввода информации?
6. Какая функция используется в Си для вывода информации?
7. Какой тип данных Си соответствует спецификатору «%d»?
8. Какой тип данных Си соответствует спецификатору «%f»?
9. Переменная j описана в программе следующим образом: `int j`; Запишите функцию `scanf` для считывания значения в переменную j .
10. Переменная k описана в программе следующим образом: `float k`; Запишите функцию `printf` для вывода значения переменной k .

Пример выполнения индивидуального варианта

Вариант 1. Ввести с клавиатуры целое число x . Вывести на экран значение функции $x^2 + 3.1x + 7.5$ и сообщение вида: «Программу выполнил ФИО».

В таблице 1.1 представлена программа, реализующая индивидуальное задание.

Таблица 1.1 – Этапы выполнения лабораторной работы

№	Название этапа	Описание результатов выполнения этапа
2	Создание проекта	Запустить Dev-C++, создать новый проект, дополнить код программы вызовом функции <code>system ("chcp 1251");</code> – смена кодировки страницы.
3	Реализация программы	
3.1	Описание переменных	<code>int x;</code> <code>float y;</code>
3.2	Ввод данных с клавиатуры	<code>printf("Введите значение переменной x: ");</code> <code>scanf("%d",&x);</code>
3.3	Вычисление значения функции	<code>y = x*x+3.1*x + 7.5;</code>
4	Вывод результата	<code>printf("Значение функции: %7.2f\n",y);</code>
5	Вывод личных данных	<code>printf("Программу выполнил Иванов Андрей Сергеевич\n");</code>

После набора представленной выше программы в шаблоне функции `main ()` (рис. 1.1) выполните компиляцию проекта.

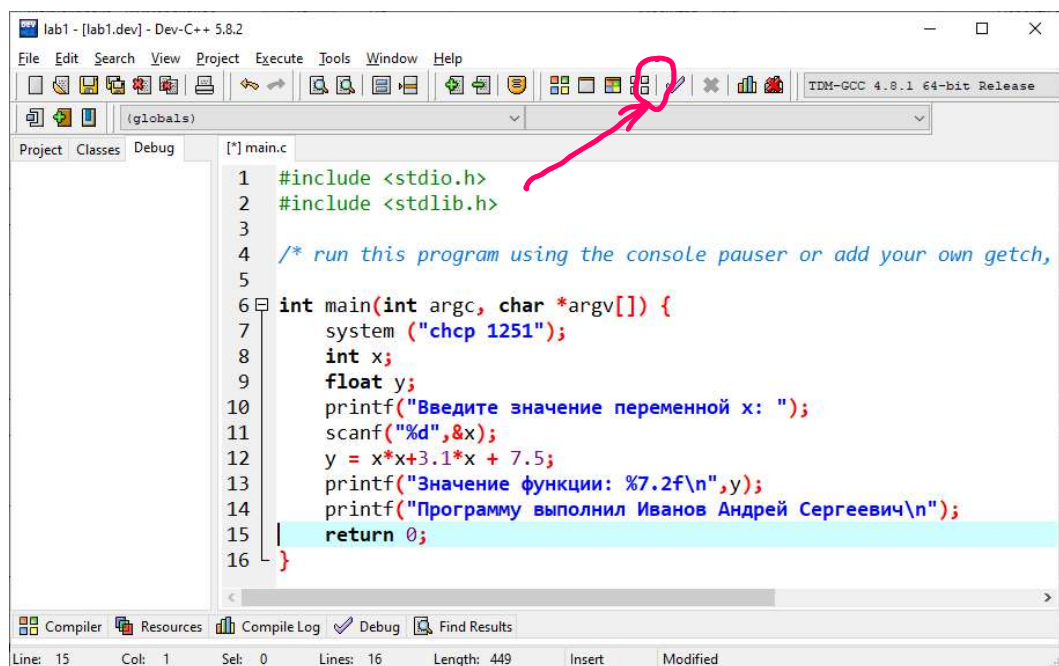


Рисунок 1.1 – Проект выполнения индивидуального задания

Определите имя файла, содержащего функцию `main()`. В рассматриваемом примере имя файла определено как `lab1` (рис. 1.2).

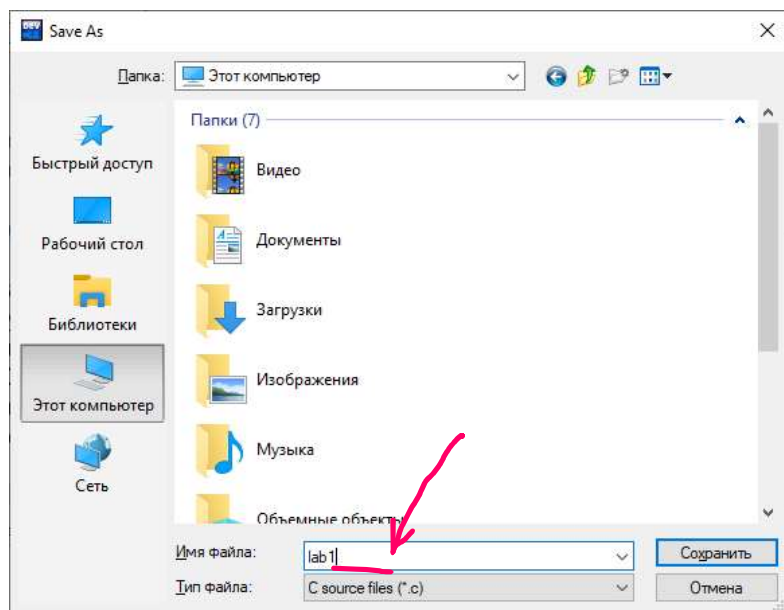


Рисунок 1.2 – Определение имени файла

Если этап компиляции прошел успешно, программа автоматически выполнится. Для смены кодировки страницы зайдите в свойства консольного окна, выберите вкладку «Шрифт» и выберите шрифт `Lucida Console`.

Результат работы программы представлен на рисунке 1.3.

```
C:\Users\perry\Downloads\lab1.exe
Текущая кодовая страница: 1251
Введите значение переменной x: 12
Значение функции: 188.70
Программу выполнил Иванов Андрей Сергеевич

-----
Process exited after 15.93 seconds with return value 0
для продолжения нажмите любую клавишу . . .
```

Рисунок 1.3 – Тестирование программы при $x = 12$

При выполнении лабораторной работы в консольном окне выведено полученное значение переменной и личные данные студента, выполняющего работу.

1.3 Лабораторная работа «Условные алгоритмы. Проверка ошибок ввода данных»

Цель работы: ознакомиться с возможностями функции `scanf()`. Научиться составлять условные алгоритмы на примере алгоритма проверки ошибок ввода данных. Реализовать алгоритм на языке Си.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе: для выполнения лабораторной работы необходимо изучить теоретический материал, изложенный в [1]. Основные конструкции структурного программирования рассматриваются в главе 1 пособия (стр. 13 – 15). Описание синтаксиса конструкции проверки условия в языке Си (стр. 87 – 90) в главе 6 пособия. Возможности функции `scanf` для организации проверки корректности ввода данных описаны на стр. 80 – 81.

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. По индивидуальному варианту определить типы и значения данных, являющиеся некорректными для задачи.
3. Составить и записать алгоритм решения задачи.
4. Составить программу, реализующую алгоритм:
 - 4.1) описать входные и выходные данные;
 - 4.2) ввести данные с клавиатуры;
 - 4.3) проверить входные данные;
 - 4.4) вычислить значение функции;
 - 4.5) вывести полученное значение на экран;
 - 4.6) вывести личные данные;
 - 4.7) выполнить компиляцию проекта.
5. Защитить работу.

Пример выполнения индивидуального варианта

Вариант 1. Составить и записать алгоритм решения индивидуального задания с проверкой корректности данных. По составленному алгоритму написать программу на языке Си.

Даны x, y, z . Вычислить a, b , если $a = \left(\frac{2y}{z-x} - |x|\right) \left(\sqrt[4]{x - \frac{y}{\sqrt{x}}}\right)$, $b = a - \frac{x}{2a} + \frac{x^2}{a-z}$. Значения x, y, z вводить с клавиатуры.

В таблице 1.2 описано поэтапное выполнение лабораторной работы.

Таблица 1.2 – Этапы выполнения лабораторной работы

№	Название этапа	Описание результата выполнения этапа
2.	Определение некорректных данных	Символьные данные; $z - x = 0$ – ошибка деления на 0; $x \leq 0$ – ошибка извлечения квадратного корня, ошибка деления на 0; $x - \frac{y}{\sqrt{x}} < 0$ – ошибка извлечения корня четвертой степени; $a = 0$ – ошибка деления на 0; $a = z$ – ошибка деления на 0.
3.	Алгоритм	<u>алг</u> Лабораторная работа 2 <u>нач</u> <u>ввод</u> x, y, z <u>если</u> x или y или z – не число, <u>то</u> <u>рез</u> “Введены не числа” <u>если</u> $z-x=0$ или $x = 0$, <u>то</u> <u>рез</u> “Ошибка деления на 0” <u>если</u> $x - \frac{y}{\sqrt{x}} < 0$ или $x < 0$ <u>то</u> <u>рез</u> “Ошибка извлечения корня” $a = \left(\frac{2y}{z-x} - x \right) \left(\sqrt[4]{x - \frac{y}{\sqrt{x}}}\right)$ <u>рез</u> a <u>если</u> $a = 0$ или $a = z$ <u>то</u> <u>рез</u> “Ошибка деления на 0” $b = a - \frac{x}{2a} + \frac{x^2}{a-z}$ <u>рез</u> b <u>конец</u>
4.	Составление программы	Для использования математических функций и констант подключите библиотеку математических функций: <code>#include <math.h></code>
4.6.	Вывод личных данных	<code>printf(“Программу выполнил Иванов Андрей Сергеевич\n”);</code>
4.1.	Описание переменных	<code>float x, y, z, a, b;</code>
4.2.	Ввод данных с клавиатуры	<code>printf(“Введите значения переменных: ”);</code> <code>int m = scanf(“%f%f%f”, &x, &y, &z);</code>
4.3.	Проверка входных данных	<code>if (m!=3) {</code> <code>printf (“Введены не числа\n”);</code> <code>system(“pause”);</code> <code>return 0;}</code> <code>if (x==0 z-x==0) {</code>

		<pre>printf ("Ошибка деления на 0\n"); system("pause"); return 0;} float temp = x - y/sqrt(x); if (temp<0 x<0) { printf("Ошибка извлечения корня\n"); system("pause"); return 0;} </pre>
4.4	Вычисление значения функции	<pre>a = 2*y/(z-x)-fabs(x); a = a*pow(temp,1/4.); </pre>
4.5	Вывод результата	<pre>printf("Значение a = %9.3f\n",a); </pre>
4.3.	Проверка входных данных	<pre>if (a==0 a==z) { printf ("Ошибка деления на 0\n"); system("pause"); return 0;} </pre>
4.4	Вычисление значения функции	<pre>b = a - x/(2*a)+ x*x/(a-z); </pre>
4.5	Вывод результата	<pre>printf("Значение b = %9.3f\n",b); </pre>

Контрольные вопросы

1. Что возвращает функция *scanf()*?
2. Запишите функцию *scanf()* для ввода трех переменных.
3. Что Вы понимаете под некорректными данными?
4. Какие данные будут некорректными для решения следующей задачи – «Даны длины трех сторон треугольника. Найдите площадь треугольника.»
5. Как в Си реализована условная конструкция структурного программирования?
6. Опишите синтаксис конструкции *if else* языка Си.
7. Какое значение примет переменная *m* после выполнения следующего фрагмента программы:

```
...
float i;
int j;
int m = scanf("%f%d",&i, &j);

```

...
если с клавиатуры были введены значения 3 2?

8. Какое значение примет переменная *m* после выполнения следующего фрагмента программы:

```
...
float i;
int j;
int m = scanf("%f%d",&i, &j);

```

...
если с клавиатуры были введены значения 3 d?

9. Какое значение примет переменная *x* после выполнения следующего фрагмента программы, представленного ниже?

```
...
int x = 10;
int k = 12,z = 74;
if (k<z) x = 1; else x = 0; ...

```

1.4 Лабораторная работа «Условные алгоритмы. Проверка условий. Геометрия на плоскости»

Цель работы: закрепление навыков построения разветвляющихся алгоритмов.

Форма проведения: выполнение индивидуального задания.

Проверка расположения точки с координатами (x, y) относительно прямой: пусть уравнение прямой задано в каноническом виде $y = ax + b$. Тогда все точки, лежащие на линии прямой (рис. 1.4), подчиняются условию $y = ax + b$. На рисунке это условие выполняется для точки с координатами (x_3, y_3) . Все точки, лежащие левее линии прямой, подчиняются условию $y < ax + b$, это условие выполняется для точки с координатами (x_1, y_1) . Все точки, лежащие правее линии прямой, подчиняются условию $y > ax + b$. Это условие является истинным для точки с координатами (x_2, y_2) . Тогда для выбранных трех точек являются истинными условия: $y_3 = ax_3 + b$; $y_2 > ax_2 + b$; $y_1 < ax_1 + b$.

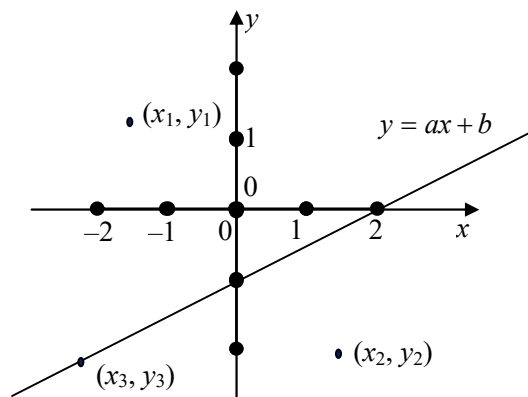


Рисунок 1.4 – Расположение точки относительно прямой

Для прямой, изображенной на рис. 4, составим уравнение прямой по двум заданным точкам: прямая проходит через точки с координатами $(0, -1)$ и $(2, 0)$. Найдем коэффициенты уравнения a и b . Для этого решим систему уравнений:

$$\begin{cases} -1 = a \cdot 0 + b \\ 0 = a \cdot 2 + b \end{cases} \Rightarrow \begin{cases} b = -1 \\ a = \frac{-b}{2} \end{cases} \Rightarrow \begin{cases} b = -1 \\ a = 0.5 \end{cases} \Rightarrow y = 0.5x - 1.$$

Таким образом, проверить местоположение произвольной точки с координатами (x, y) можно, написав следующий код:

```
...
if (y < 0.5*x - 1)
printf("Точка расположена левее прямой");
else if (y > 0.5*x - 1)
printf("Точка расположена правее прямой");
else printf("Точка расположена на прямой");
...
```

Проверка расположения точки относительно окружности с заданным центром известного радиуса: каноническое уравнение окружности выглядит следующим образом:

$$R^2 = (x - x_1)^2 + (y - y_1)^2, \tag{1}$$

где R – радиус окружности,
 (x_1, y_1) – координаты центра окружности.

Тогда (рис. 1.5) для точки с координатами (x_4, y_4) , выполняется равенство:

$$R^2 = (x_4 - x_1)^2 + (y_4 - y_1)^2.$$

Выражение (1) истинно для всех точек, лежащих на линии окружности. Для точки с координатами (x_2, y_2) и для всех точек, лежащих за окружностью, выполняется неравенство:

$$R^2 < (x_2 - x_1)^2 + (y_2 - y_1)^2.$$

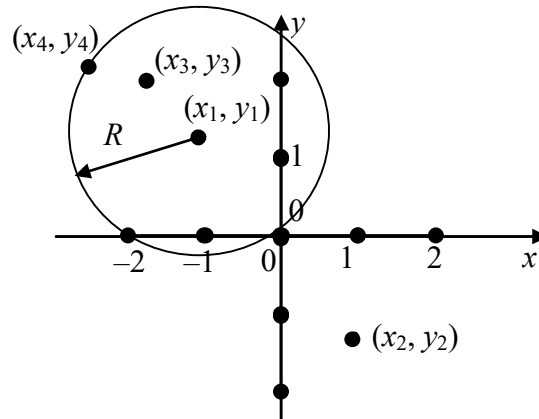


Рисунок 1.5 – Расположение точки относительно окружности

Из этого неравенства следует, что радиус R окружности меньше радиуса окружности с центром в точке (x_1, y_1) , на которой лежит точка с координатами (x_2, y_2) . Соответственно, для точки с координатами (x_3, y_3) выполняется неравенство: $R^2 > (x_3 - x_1)^2 + (y_3 - y_1)^2$.

То есть радиус R окружности, на которой лежит точка с координатами (x_3, y_3) больше радиуса окружности с центром в точке (x_1, y_1) .

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Определить условия вхождения точки в заданную область.
3. Составить и записать алгоритм решения задачи.
4. Составить программу, реализующую алгоритм:
 - 4.1. описать входные и выходные данные;
 - 4.2. ввести данные с клавиатуры;
 - 4.3. проверить входные данные;
 - 4.4. проверить условие вхождения точки в заданную область;
 - 4.5. вывести результат проверки на экран;
 - 4.6. вывести личные данные.
5. Выполнить компиляцию проекта.
6. Защитить работу.

Примеры проверки вхождения точки с заданными координатами в фигуру

Пример 1. Напишите алгоритм, проверяющий вхождение точки в область, изображенную на рисунке 1.6.

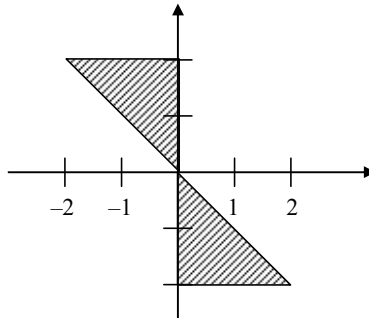


Рисунок 1.6 – Пример 1

Построим условия вхождения точки в заданную область:

Уравнение прямой, на которой лежат гипотенузы прямоугольных треугольников, образующих фигуру: $y = -x$.

Разобьем фигуру на две части. Точка будет считаться принадлежащей фигуре, если она попадет в первую или вторую часть.

Первую (верхнюю часть) можно ограничить следующими условиями: $(y \geq -x)$ и $(x \leq 0)$ и $(y \leq 2)$.

Первое условие описывает гипотенузу, второе и третье условие описывают катеты. Условия связаны между собой связками **И** (логическое умножение).

Вторую (нижнюю часть) можно ограничить условиями:

$(y \leq -x)$ и $(x \geq 0)$ и $(y \geq -2)$.

Общее условие для двух частей будет выглядеть следующим образом:

если $(y \geq -x)$ **и** $(x \leq 0)$ **и** $(y \leq 2)$ **или**
 $(y \leq -x)$ **и** $(x \geq 0)$ **и** $(y \geq -2)$,

то «Точка принадлежит заданной области»,

иначе «Точка не принадлежит заданной области».

Тогда алгоритм проверки вхождения точки с заданными координатами будет выглядеть следующим образом:

алг Пример 1 нач

вещ x, y

ввод x, y

если $(y \geq -x)$ **и** $(x \leq 0)$ **и** $(y \leq 2)$ **или**
 $(y \leq -x)$ **и** $(x \geq 0)$ **и** $(y \geq -2)$,

то «Точка принадлежит заданной области»,

иначе «Точка не принадлежит заданной области»

кон

Пример 2. Рассмотрим еще один пример, заданная область изображена на рисунке 1.7. В этом случае: уравнение прямой $y = x$; уравнение окружности $l = (x - 1)^2 + (y - 1)^2$.

Ограниченная область находится:

- правее прямой ($y < x$);
- внутри окружности ($l > (x - 1)^2 + (y - 1)^2$).

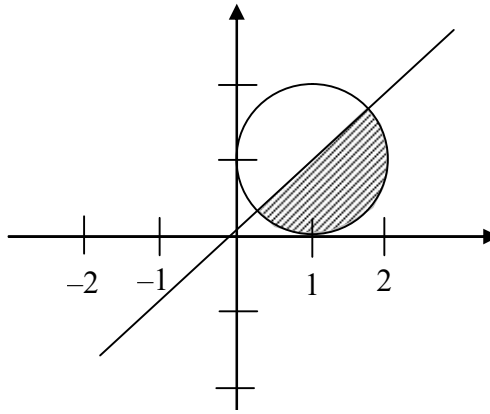


Рисунок 1.7 – Пример 2

Тогда общее условие будет выглядеть следующим образом:

если $y < x$ **и** $1 > (x - 1)^2 + (y - 1)^2$,

то «Точка принадлежит заданной области»,

иначе «Точка не принадлежит заданной области».

В этом случае алгоритм проверки вхождения точки с заданными координатами будет выглядеть следующим образом:

алг Пример 2 **нач**

вещ x, y

ввод x, y

если $y < x$ **и** $1 > (x - 1)^2 + (y - 1)^2$,

то «Точка принадлежит заданной области»,

иначе «Точка не принадлежит заданной области»

кон

Контрольные вопросы

1. Как должно быть построено условие нахождения точки с заданными координатами относительно прямой?
2. Как определяются условия расположения точки с заданными координатами относительно окружности?
3. Какие логические операторы используются для построения условий?
4. Какие средства языка программирования Си позволяют реализовать разветвляющиеся алгоритмы?

1.5 Лабораторная работа «Итерационные алгоритмы. Вычисление суммы бесконечного ряда»

Цель работы: закрепление навыков программирования итерационных процессов.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе: для выполнения лабораторной работы необходимо изучить теоретический материал, изложенный в [1]. Основные конструкции структурного программирования рассматриваются в главе 1 пособия (стр. 13 – 15). Описание синтаксиса конструкций циклов в языке Си (стр. 91 – 95) в главе 6 пособия. Математические основы вычисления суммы бесконечного ряда и логика процесса вычисления суммы бесконечного ряда описана на стр. 95 – 97.

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Если необходимо, преобразовать исходную формулу ряда в рекуррентную формулу.
3. Написать и протестировать программу вычисления суммы бесконечного ряда.
4. Защитить работу.

Контрольные вопросы

1. Почему при программировании алгоритма вычисления бесконечного ряда рекомендуется вывести рекуррентную формулу?
2. К каким ошибкам может привести вычисление степени аргумента и факториала при программировании бесконечного ряда?
3. Что такое точность вычисления бесконечного ряда?
4. Какой прием алгоритмизации был использован для программирования $(-1)^n$?
5. Как зависит количество итераций алгоритма от заданной точности вычислений?
6. Как зависит количество итераций алгоритма от заданного значения алгоритма?

1.6 Лабораторная работа «Обработка массивов. Поиск»

Цель работы: закрепление навыков обработки статического одномерного массива.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе: для выполнения лабораторной работы необходимо изучить теоретический материал, изложенный в [1]. Справочная информация об организации работы с массивами описана в главе 8 пособия (стр. 114 – 120). Графические способы представления алгоритмов описываются в главе 1 (стр. 15 – 20).

Порядок выполнения работы

1. Получить индивидуальное задание.
2. Составить алгоритм решения задания и реализовать его графическое представление (блок-диаграмма, диаграмма Насси-Шнайдермана).
3. Составить программу на языке Си.
4. Выполнить отладку и тестирование программы.
5. Защитить работу.

Пример выполнения индивидуального задания

Вводится последовательность из n целых чисел. Сохранить все введенные значения в массиве и найти сумму всех отрицательных чисел (*индивидуальное задание*).

Блок-диаграмма задачи представлена на рисунке 1.8.

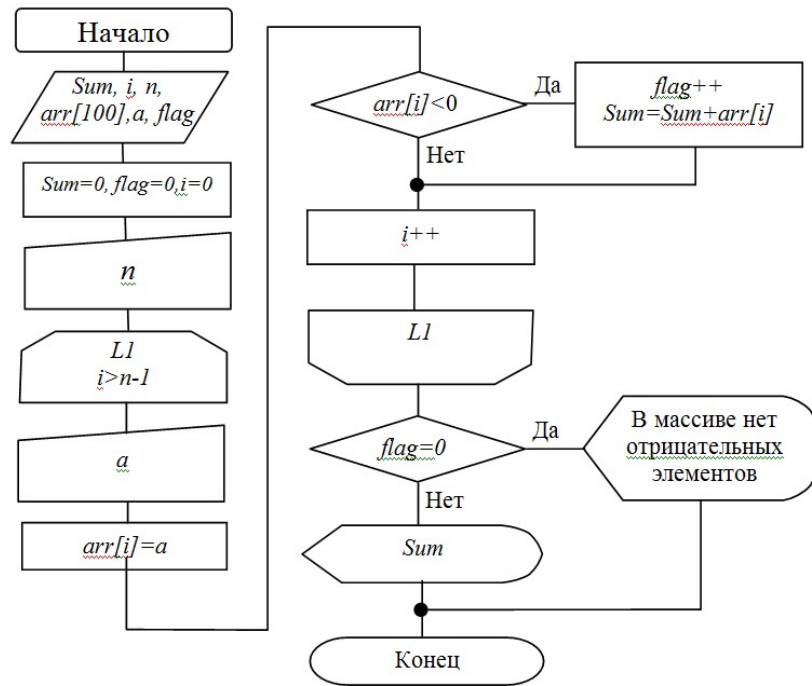


Рисунок 1.8 – Алгоритм поиска суммы отрицательных элементов

Реализация задания на языке Си может быть выполнена следующим образом:

```
int main(int argc, char *argv[])
{system("chcp 1251");
int Sum=0, a, n, arr[100];
int flag = 0;
printf("Введите количество членов последовательности не более 100 ");
scanf("%d",&n);
for(int i=0; i<n; i++){
    printf("Введите значение члена последовательности");
    scanf("%d", &a);
    arr[i]=a;
    if (arr[i]<0) {Sum+=arr[i]; flag++;}
}
if (!flag) printf("В массиве нет отрицательных элементов\n");
else
printf("Значение суммы отрицательный членов
последовательности равна %6d", Sum);}
```

Контрольные вопросы

1. Почему нумерация элементов массива в языке Си начинается с 0?
2. Как связаны указатели и массивы?
3. Как располагаются в памяти компьютера элементы массива?
4. Что произойдет если в программе на языке Си обратиться к несуществующему элементу массива?
5. Правильно ли реализован алгоритм поиска минимального элемента массива, представленный ниже?

```
int x[10];
int i,min;
for(i=0;i<n-1;i++)
    if(x[i]<x[i+1]) min = x[i]; else min = x[i+1];
```
6. Приведите пример массива, для которых алгоритм будет работать верно.

1.7 Обработка массивов. Динамические массивы

Цель работы: изучение способов работы с динамическими одномерными массивами; реализация простых алгоритмов сортировки и поиска; организация отладки программы с помощью встроенного отладчика.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе:

При решении задач очень часто размер массива является переменной величиной. В этом случае используются динамические массивы. Инициализация таких массивов выполняется в два этапа. Первый этап – выделение памяти, второй этап – инициализация значений элементов массива. Элементы массива могут вводиться с клавиатуры, так как это показано в примере:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    system("chcp 1251");
    // Описание указателя на целое число
    int *x;
    int n, i;
    printf("Введите размерность массива: ");
    scanf("%d", &n);
    // выделение памяти
    x = (int*)malloc(sizeof(int)*n);
    for(i=0; i<n; i++){
        printf("x[%d] -> ", i);
    // чтение элементов с клавиатуры
        scanf("%d", &x[i]);
    }
    // очистка экрана
    system("cls");
    // печать элементов массива
    for(i=0; i<n; i++) printf("%5d", x[i]);
    // освобождение памяти
    free(x);
    printf("\n");
    return 0;}

```

Помимо функции malloc в языке Си реализована функция calloc, синтаксис которой представлен ниже.

```
void * calloc( size_t number, size_t size );
<stdlib.h>
```

Функция calloc выделяет блок памяти для массива размером – num элементов, каждый из которых занимает size байт, и инициализирует все выделенные биты нулями.

В результате выделяется блок памяти размером number * size байт, причём весь блок заполнен нулями.

Параметры:

number – количество элементов массива, под который выделяется память;
size – размер одного элемента в байтах.

Возвращаемое значение – указатель на выделенный блок памяти. Тип данных, на который ссылается указатель всегда `void*`, поэтому это тип данных может быть приведен к желаемому типу данных.

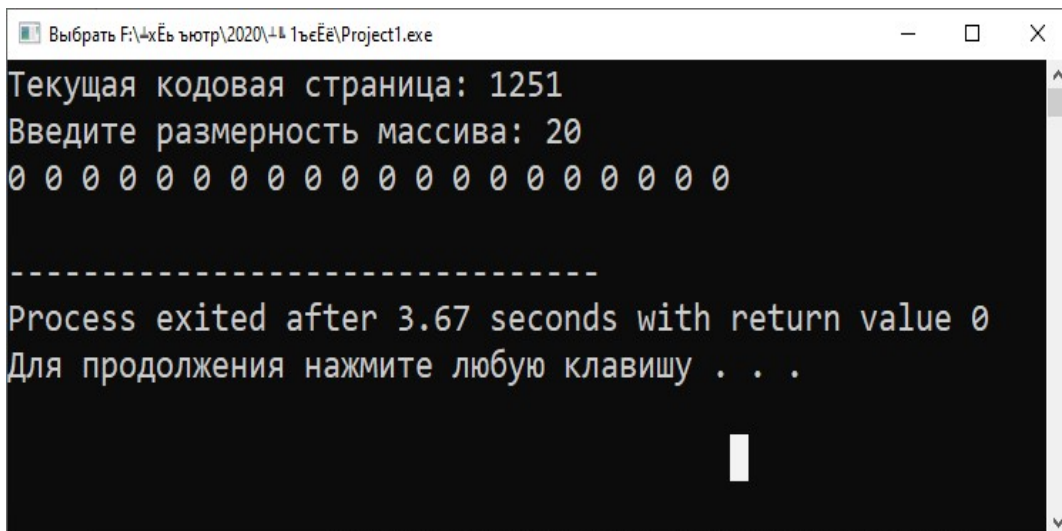
Если функции не удалось выделить требуемый блок памяти, возвращается нулевой указатель.

Пример использования функции `calloc` приведен ниже.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    system("chcp 1251");
    // Описание указателя на целое число
    int *x;
    int n, i;
    printf("Введите размерность массива: ");
    scanf("%d", &n);
    // выделение памяти
    x = (int*)calloc(n, sizeof(int));
    for(i=0; i<n; i++)
        printf("%d ", x[i]);
    // освобождение памяти
    free(x);
    printf("\n");
    return 0;}

```

Результат работы программы продемонстрирован на рисунке 1.9.



```
Выбрать F:\ХЕЬ юотр\2020\11 1гееЕ\Project1.exe
Текущая кодовая страница: 1251
Введите размерность массива: 20
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-----
Process exited after 3.67 seconds with return value 0
Для продолжения нажмите любую клавишу . . .

```

Рисунок 1.9 - Результат работы функции `calloc`

При запуске программы видно, что, хотя инициализация элементов массива не выполнялась элементы массива, тем не менее равны 0.

Наряду с двумя описанными выше функциями в языке Си определена еще одна функция для работы с памятью – функция `realloc`.

```
void * realloc( void * ptrmem, size_t size );
<stdlib.h>
```

Функция `realloc` выполняет перераспределение блоков памяти.

Размер блока памяти, на который ссылается параметр *ptrmem* изменяется на *size* байтов. Блок памяти может уменьшаться или увеличиваться в размере.

Эта функция может перемещать блок памяти на новое место, в этом случае функция возвращает указатель на новое место в памяти. Содержание блока памяти сохраняется даже если новый блок имеет меньший размер, чем старый. Отбрасываются только те данные, которые не вместились в новый блок. Если новое значение *size* больше старого, то содержимое вновь выделенной памяти будет неопределенным.

В случае, если *ptrmem* равен NULL, функция ведет себя именно так, как функция *malloc*, т. е. выделяет память и возвращает указатель на этот участок памяти.

В случае, если *size* равен 0, ранее выделенная память будет освобождена, как если бы была вызвана функция *free*, и возвращается нулевой указатель.

Параметры:

ptrmem – указатель на блок ранее выделенной памяти функциями *malloc*, *calloc* или *realloc* для перемещения в новое место. Если этот параметр – NULL, просто выделяется новый блок, и функция возвращает на него указатель.

size – новый размер, в байтах, выделяемого блока памяти. Если *size* равно 0, ранее выделенная память освобождается, и функция возвращает нулевой указатель, *ptrmem* устанавливается в 0.

Возвращаемое значение

Указатель на перераспределенный блок памяти, который может быть либо таким же, как аргумент *ptrmem* или ссылаться на новое место.

Тип данных возвращаемого значения всегда *void**, который может быть приведен к любому другому.

Если функции не удалось выделить требуемый блок памяти, возвращается нулевой указатель, и блок памяти, на который указывает аргумент *ptr* остается неизменным.

Функция *realloc* может быть использована, например, для решения следующей задачи: *пользователь вводит с клавиатуры положительные целые числа, при этом количество вводимых чисел заранее не известно, но не более 100. Ввод чисел заканчивается если пользователь ввел отрицательное число или 100 чисел подряд. Все введенные числа нужно сохранить в массиве, размерность которого должна быть равна количеству введенных чисел.*

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    system("chcp 1251");
    // Описание указателя на целое число
    int *x;
    int n = 0, i;
    // выделение памяти под 100 элементов массива
    x = (int*)calloc(100, sizeof(int));
    printf("Вводите числа (ввод будет прекращен \
        \n после ввода первого отрицательного числа \
        \n или после ввода 100 чисел подряд ): ");
    do{
        scanf("%d",&x[n]);
        if (x[n]<0) break;
        n++;
    }while(n<100);
    //перераспределение памяти
```

```

x = realloc(x,n*sizeof(int));
if (x==NULL) printf("Массив пуст");
else {
    printf("Получен массив: \n");
    for(i=0;i<n;i++)
        printf("%d ",x[i]);
}
// освобождение памяти
free(x);
printf("\n");
return 0;}

```

На рисунках 1.10,1.11 представлены результаты работы программы:

```

F:\xЕё\ютр\2020\+Е 1ъЕЕ\main1.exe
Текущая кодовая страница: 1251
Вводите числа (ввод будет прекращен
  после ввода первого отрицательного числа
  или после ввода 100 чисел подряд ): -20
Массив пуст

-----
Process exited after 11.94 seconds with return
  value 0
Для продолжения нажмите любую клавишу . . .

```

Рисунок 1.10 – После работы программы получен пустой массив

```
F:\ХЕБь Ыютр\2020\11 1ьсЕё\main1.exe
Текущая кодовая страница: 1251
Вводите числа (ввод будет прекращен
  после ввода первого отрицательного числа
  или после ввода 100 чисел подряд ): 20
30
15
-10
Получен массив:
20 30 15

-----
Process exited after 11.71 seconds with return
  value 0
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1.11 – После работы программы получен массив из трех элементов

Результаты выполнения программы, приведенные выше показывают, что программа работает корректно: в первом случае массив оказался пуст, потому что пользователь ввел первое отрицательное число; во втором случае отрицательное число было введено четвертым по счету, первые три положительных числа сохранились в массиве.

Следующий рассмотренный пример организует корректную работу с памятью, в случае удаления из массива заданных элементов. Пусть необходимо решить следующую задачу: дан массив из n элементов (размерность массива задается с клавиатуры, элементы массива задаются случайным образом). Написать программу, которая удаляет из массива элемент с заданным номером k . Номер k вводится с клавиатуры.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    system("chcp 1251");
    // Описание указателя на целое число
    int *x;
    int n,i,k;
    printf("Вводите размерность массива: ");
    scanf("%d",&n);
    // выделение памяти
    x = (int*)malloc(sizeof(int)*n);
    for(i=0;i<n;i++){
        x[i] = rand()%100;
        printf("%d ",x[i]);}
}
```

```

printf("\n Введите номер удаляемого элемента: ");
scanf("%d",&k);
// сдвиг влево элементов массива, начиная с k+1
for(i=k+1;i<n;i++){
    x[i-1] = x[i];
}
// уменьшение размерности
n--;
// перераспределение памяти
x = realloc(x,sizeof(int)*n);
printf("Получен массив: \n");
for(i=0;i<n;i++){
    printf("%d ",x[i]);}
// освобождение памяти
free(x);
printf("\n");
return 0;}

```

Результат работы программы представлен на рисунке 1.12.

```

F:\ХЕБ\ютр\2020\14 1геЕЕ\main.exe
Текущая кодовая страница: 1251
Введите размерность массива: 10
41 67 34 0 69 24 78 58 62 64
Введите номер удаляемого элемента: 3
Получен массив:
41 67 34 69 24 78 58 62 64
-----
Process exited after 7.164 seconds with return value 0
Для продолжения нажмите любую клавишу . . .

```

Рисунок 1.12 – Тестирование программы

На приведенных тестовых данных сдвигаются влево все элементы, начиная с элемента с номером 4. В итоге элемент $x[3] = 0$ оказался удаленным из массива.

Требования к выполнению задания

Программа должна быть написана с использованием динамических массивов (должны быть использованы функции `malloc` или `calloc`). При решении пункта задания должно быть выполнено перераспределение памяти, необходимой для хранения массива (должна быть использована функция `realloc`).

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Разработать алгоритм, реализующий задание индивидуального варианта.
3. Написать программу, реализующую решение задания индивидуального варианта.
4. Отладить и протестировать программу.
5. Защитить программу.

Контрольные вопросы

1. Всегда ли эффективно использование динамической памяти в программах?
2. Как создать динамический массив в языке программирования Си?
3. Почему ошибки при работе с динамической памятью относят к опасным?
4. Почему выделенную ранее динамическую память следует освобождать после использования?
5. Почему к нулевому указателю нельзя применить операцию разыменования?
6. Какие функции языка программирования Си используются для выделения памяти?

1.8 Лабораторная работа «Обработка матриц. Проектирование пользовательских функций»

Цель работы: изучение способов работы с динамическими двумерными массивами; реализация простых алгоритмов сортировки и поиска; организация отладки программы с помощью встроенного отладчика; закрепление навыков разработки функций.

Форма проведения: выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе: для выполнения лабораторной работы необходимо изучить теоретический материал, изложенный в [1]. В главе 8 пособия изложена справочная информация об организации работы с двумерными массивами в языке Си (стр. 114 – 120), способы сортировки наборов данных рассмотрены (стр. 121 – 123). Руководство по использованию встроенного отладчика в IDE описано в главе 2 (стр. 37 – 42). В главе 7 пособия описаны основные моменты организации функций на языке Си (стр. 103 – 109).

Требования к выполнению лабораторной работы: программа должна иметь дружелюбный интерфейс; информация, которая будет выведена на экран, должна показать пользователю, что задание решено верно. Например, если в задании необходимо изменить порядок строк матрицы, упорядочив их по значению минимального элемента, то вывод должен быть организован таким образом, как показано на рисунке 1.13.

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Изучить теоретические аспекты лабораторной работы.
3. Изучить работу отладчика в среде Dev-C++.
4. Разработать и реализовать на языке Си алгоритм решения предложенных задач.
5. Защитить работу, используя средства отладчика на тестовом примере с матрицей [5x5].

```
G:\Хьюэйт\2018\...main51.exe
Текущая кодовая страница: 1251
Введите количество строк: 6
Введите количество столбцов: 4
 8 37 66 73 min = 8
 4  5 87 22 min = 4
87  5 15 61 min = 5
92 21 70  6 min = 6
 7 73 41 45 min = 7
73 95 63 37 min = 37

 4  5 87 22 min = 4
87  5 15 61 min = 5
92 21 70  6 min = 6
 7 73 41 45 min = 7
 8 37 66 73 min = 8
73 95 63 37 min = 37

-----
Process exited after 34.31 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1.13 – Пример организации вывода

Контрольные вопросы

1. Опишите механизм выделения памяти для двумерного массива.
2. Какими способами можно изменить порядок строк в матрице?
3. Как представить двумерный массив в виде одномерного массива?
4. Что определяет первый индекс двумерного массива?
5. Опишите алгоритм транспонирования матрицы.
6. Где в описании функции указывается тип возвращаемого значения?
7. В каком случае необходимо использовать передачу параметров по ссылке?
8. Какой оператор языка Си заканчивает выполнение функции?
9. Обязательно ли функция должна возвращать значение?
10. Как в языке Си можно реализовать передачу функции параметром?

1.9 Лабораторная работа «Текстовые файлы»

Цель работы – закрепление навыков работы с текстовыми файлами.

Форма проведения – выполнение индивидуального задания.

Рекомендации по подготовке к лабораторной работе. Для выполнения задания необходимо изучить теоретический материал, изложенный в [1]. В главе 9 пособия описаны стандартные функции языка Си для работы с текстовыми файлами (стр. 144 – 155). Изучите режимы открытия файла, функции для работы с текстовыми файлами. Попробуйте выполнить примеры работы с текстовыми файлами, приведенные в пособии.

Дополнительную информацию по работе с текстовыми файлами можно получить, изучив пособие [2]. В главе 7 (стр. 283 - 290) определяются стандартные потоки ввода-вывода в языке программирования Си. В разделе 7.1.3 описываются способы работы с файлами на диске.

Порядок выполнения работы

1. Получите индивидуальное задание на лабораторную работу.

2. Создайте в папке проекта текстовый файл, заполните его согласно варианту.
3. Напишите программу, считывающую текстовый файл и изменяющую его.
4. Отладьте программу.
5. Защитите результаты лабораторной работы.

Контрольные вопросы

1. Дайте определение файла в языке Си.
2. Что обозначает термин «поточный ввод-вывод»?
3. Для чего служит предопределенная структура FILE?
4. Укажите различия между именем файла на диске и указателем на структуру FILE.
5. Какая функция открывает поток в текстовом режиме?
6. Запишите функцию fscanf для чтения переменной z типа float из потока m.

1.10 Лабораторная работа «Синтаксис ООП»

Цель работы – ознакомиться с основными принципами объектно-ориентированного программирования и синтаксисом C++ для создания объектно-ориентированных приложений на примере класса Vector.

Форма проведения – выполнение индивидуального задания.

Синтаксис C++ для работы с классами. В языке программирования C++ основным элементом объектно-ориентированных приложений является класс. Синтаксис описания класса:

```
class <имя класса>
{
    список членов (описание полей и методов)
};
```

Члены класса можно разделить на две группы. Поля класса – данные, каким – то образом идентифицирующие объект класса, либо хранящиеся в объекте класса. Методы класса – функции, которые могут использовать поля класса и внешние данные.

Каждый член класса имеет атрибут доступа, с их помощью определяется “зона видимости” для члена класса. Всего таких атрибутов 3:

- **public** – член класса может использоваться любой функцией. Как правило, этот атрибут используется для функций-членов классов (методов); т.е. такой член видим и вне класса.
- **private** – член класса может использоваться только функциями-членами класса и функциями-друзьями класса. Как правило используется для данных класса (полей), т.е. скрывает данные внутри класса.
- **protected** – член класса может использоваться функциями-членами класса и функциями-друзьями класса, а также членами классов, для которых этот класс является базовым (предком).

Существуют специальные функции-члены, которые отвечают за создание объектов класса, их копирование и удаление. Это так называемые конструкторы и деструкторы. Конструкторы – это функции-члены класса, основная цель которых инициализация объекта или распределение памяти для хранения объекта. Конструктор имеет то же имя, что и класс. Так же, как и любая функция, конструктор может иметь и не иметь параметров. Конструктор без параметров называется конструктором по умолчанию. Конструктора может не быть, тогда C++ генерирует собственный конструктор по умолчанию.

Деструктор разрушает созданный объект. У деструктора нет параметров. Деструктор вызывается, если закончена часть программы, в которой видим объект. Деструктора носит имя класса с лидирующим знаком «~» - тильда. Если деструктор не описан в классе явно, то используется деструктор по умолчанию. Деструктор не может перегружаться.

Порядок выполнения работы

1. Самостоятельно рассмотрите реализацию методов класса VECTOR. Обратите внимание на синтаксис описания методов.
2. Получите вариант индивидуального задания у преподавателя.
3. Для компиляции программы создайте проект. В проект должны быть включены 5 файлов:
 - Файл с функцией main() (test.cpp),
 - Файл с описанием метода Run класса MyVector (myvec.cpp),
 - Файл с описанием методов класса VECTOR (vec.cpp),
 - Файл с описанием класса VECTOR (vec.h)
 - Файл с описанием класса MyVector (myvec.h)
4. Отладьте программу.
5. Защитите выполненное задание.

Контрольные вопросы

1. Какое ключевое слово используется в языке C++ для описания класса?
2. Какой принцип ООП реализуют атрибуты доступа?
3. Перечислите поля класса VECTOR.
4. Есть ли в классе VECTOR конструктор?
5. Как можно создать объект класса VECTOR?

1.11 Лабораторная работа «Создание класса»

Цель работы – разработать класс языка C++.

Форма проведения – выполнение индивидуального задания.

Рекомендации по выполнению работы

Связь Си и Си++

При разработке языка программирования Си++ за основу был взят язык Си. Поэтому все базовые типы данных и операторы управления полностью совпадают с языком Си. Но язык Си – структурно ориентированный язык, не имеющий средств для ООП, а язык C++ включает в себя мощные средства для реализации объектно-ориентированного подхода к программированию.

Классы

Для описания объекта в C++ используется ключевое слово `class`. Синтаксис описания класса выглядит следующим образом:

```
class <имя класса>
{
<атрибут доступа> <тип> <имя поля 1>
<атрибут доступа> <тип> <имя поля 2>
...
<атрибут доступа> <тип> <имя поля k>
}
[конструктор1
конструктор 2
```

```

...]
[
<атрибут доступа> <тип> <имя метода 1>
<атрибут доступа> <тип> <имя метода 2>
...
]
[деструктор]
}

```

Опишем класс для работы с графической точкой:

```

class Point{
int x;// поле x
int y;// поле y
18
Point (int x1, int y1);// Конструктор
Point();// Конструктор
void Show();// Метод
}

```

Элементы класса

Будем называть данные, описывающие объект полями класса. В классе `Point`, рассмотренном выше, описаны два поля – целочисленные поля `x` и `y`.

Функции, принадлежащие классу, называются методами. Синтаксис аналогичен описанию функции Си. Методы могут изменять поля объекта, выводить на экран информацию, изменять объекты других классов.

В классе могут присутствовать специальные методы, которые называются конструкторами и деструкторами. Конструкторы не имеют возвращаемого значения и обязательно называются именем класса. У одного класса может быть несколько конструкторов, различающихся списками параметров. Конструкторы, как правило, используются для начальной инициализации полей класса. В классе `Point` два конструктора – 1) `Point (int x1, int y1)`, 2) `Point()`. Конструктор вызывается при описании объекта класса, либо при выделении памяти под объект. При отсутствии конструктора вызывается конструктор по умолчанию, создаваемый компилятором языка.

Деструктор служит для освобождения памяти, занятой объектом. Деструктор, так же, как и конструктор может отсутствовать. В этом случае компилятор создает деструктор по умолчанию. Деструктор носит имя класса с лидирующим знаком «~» (тильда) и не имеет параметров. Поэтому перегрузить деструктор нельзя. Рекомендуется описывать свой собственный деструктор, если в конструкторе или в методе класса выделяется память под поля класса. В этом случае в деструкторе необходимо освободить занятую память. Вызывается деструктор, когда объект, описанный в программе, заканчивает свое существование. Если в программе описан указатель на объект и под него выделялась память, деструктор вызывается при освобождении этой памяти. В классе `Point` нет деструктора.

Атрибуты доступа

C++ различает три степени видимости элементов класса. Атрибут доступа `public` говорит о том, что элемент видим извне класса. Для полей это означает возможность их использования и изменения не только методами класса, тем самым нарушается принцип инкапсуляции. Для методов – разрешение их вызова извне. Все элементы, описанные этим атрибутом, наследуются в классах-потомках.

Атрибут доступа `protected` говорит о том, что элемент нельзя использовать за пределами класса (поля нельзя изменять, а методы нельзя вызывать), однако эти элементы наследуются в классах-потомках.

Атрибут доступа `private` говорит о том, что элементы видны только внутри класса (поля и методы можно использовать только в других методах этого класса). Такие элементы не наследуются в классах-потомках.

Если для элемента класса не указан атрибут доступа, по умолчанию принимается атрибут `private`.

Исходя из всего выше написанного, изменим описание класса `Point`:

```
class Point{
private: // элементы класса, видимые лишь внутри класса
int x;
int y;
public:// элементы класса, видимые извне класса
Point (int x1, int y1);// Конструктор
Point();// Конструктор
void Show();// Метод вывода на экран
int GetX();// Метод, возвращающий значение поля x
int GetY(); // Метод, возвращающий значение поля y
void PutX(int X); // Метод, изменяющий значение поля x
void PutY(int Y); // Метод, изменяющий значение поля y
};
```

Простая программа на языке C++

Описание и инициализация объектов

Объекты в программе можно создать двумя способами. Статические объекты:

```
Point Obj;
Point Obj1(7,5);
```

При таком создании для объекта `Obj` выполняется конструктор с пустым списком параметров (обратите внимание, для конструктора в этом случае не указываются пустые скобки). Для объекта `Obj1` выполняется конструктор с параметрами. При этом для объекта `Obj` $x = 0, y = 0$. Для объекта `Obj1` $x = 7, y = 15$.

В программе можно описать и указатель на объект:

```
Point *Obj2, *Obj3;
```

Для объектов класса `Point` можно выполнить выделение памяти двумя способами.

```
Obj2 = new Point();
Obj3 = new Point(8,20);
```

В первом случае вызывается конструктор без параметров, во втором – с параметрами.

Обращение к полям и методам объекта

Если в программе описан объект существующего класса, то обращение к полям и методам класса происходит через разделитель `‘.’`.

```
Obj.x = 8; // Ошибка, поле x описано атрибутом private
Obj1.y = 12 // Ошибка, поле y описано атрибутом private
Obj.Show();// Вызов метода Show()
Obj1.PutY(12);// Вызов метода PutY()
int m=Obj.GetX();// Вызов метода GetX()
cout << "Obj.x = "<<m;
```

При обращении к полям и методам объекта, обращение к которому происходит через указатель, используется селектор `‘->’`.

```

Obj2->Show();// Вызов метода Show()
Obj3->PutX(7);// Вызов метода PutX()
int p = Obj2->GetX();// Вызов метода GetX()

```

Создание стартового класса

Выполнение программы на C++ всегда начинается с запуска функции `main()`. И можно было бы перечислить вызовы нужных методов непосредственно в `main()`, но для соблюдения принципов объектно-ориентированного программирования напишем стартовый класс, который будет содержать всего один метод:

```

#include <iostream.h>
// Описание класса Point
class Point{
private:
    int x;
    int y;
public:
    Point (int x1,int y1);
    Point();
    void Show();
    int GetX()
    {return x;}
    int GetY()
    {return y;}
    void PutX(int X);
    void PutY(int Y);
};
// Реализация методов класса Point
Point :: Point(int x1, int y1){
x = x1;
y = y1;
}
Point :: Point(){
x = 0;
y = 0;
}
void Point:: Show(){
cout << "Точка (" << x << ", " << y << ")" << endl;
}
void Point::PutX(int X){
x = X;
}
void Point::PutY(int Y){
y = Y;
}
// Описание класса Start
class Start{
public:
void Run();

```

```

};
// Реализация метода Run
void Start::Run(){
Point Obj,Obj1(7,15);// Описание объектов
Point *Obj2, *Obj3;// Описание указателей на объекты
Obj.Show();// Вызов метода Show для объектов
Obj1.Show();
Obj2 = new Point();// Выделение памяти под указатели
Obj3 = new Point(8,20);
Obj1.PutX(15);// Вызов метода, изменяющего
// значение поля x объекта
Obj1.Show();
// Проверка равенства объектов (равными
// считаются точки) у которых совпадают поля x и y.
if (Obj.GetX()==Obj2->GetX() && Obj.GetY()==Obj2->GetY())
    cout << "Объекты Obj и Obj2 равны\n" ;
    else cout << "Объекты Obj и Obj2 не равны\n";
delete Obj2;
delete Obj3;}
void main()
{
    Start St;// Описание объекта Start
    St.Run();// Вызов метода Start
}

```

Рекомендации по компоновке программ

Для облегчения ориентации в тексте программы рекомендуется использовать собственные заголовочные файлы и многофайловую компиляцию (создание проекта). В заголовочных файлах сохраняют описание класса. Называют такие файлы, как правило, по имени класса, например `Point.h`.

Реализацию методов сохраняют в файле с расширением `cpp` и называют так же по имени класса. Функция `main()` располагается в отдельном файле, в этом файле с помощью директивы `#include` подключаются заголовочные файлы. Далее создается проект, в который включаются все файлы с расширением `cpp`.

Порядок выполнения работы

1. Получить индивидуальный вариант
2. Спроектировать описание класса
3. Написать программу, реализующую решение задания индивидуального варианта
4. Отладить и протестировать программу
5. Защитить программу

Контрольные вопросы

1. Сколько полей в разработанном Вами классе?
2. Как создать статический объект класса?
3. Как создать динамический объект класса?
4. Каким атрибутом доступа описаны поля класса?

1.12 Лабораторная работа «Описание методов»

Цель работы – получение навыков объектно-ориентированного программирования.

Форма проведения – выполнение индивидуального задания.

Рекомендации по выполнению работы

Реализация методов класса

Методы класса могут быть реализованы двумя способами. Обычный способ реализации – отдельно от описания класса, с указанием принадлежности. Синтаксис такой реализации следующий:

```
<тип возвращаемого значения> <класс - хозяин> :: <имя метода>
(список параметров метода)
{ тело метода
}
```

Так же, можно описать тело метода внутри описания класса, непосредственно после заголовка метода. В этом случае, при компиляции программы, компилятор попытается сделать такие функции `inline`-функциями. `inline`-функции – это методы, код которых прописывается во все имеющиеся в программе вызовы этих методов. Таким образом, уменьшается время, потраченное на выполнение механизма вызова функции. Но ключевое слово `inline` – не инструкция компилятору, а лишь просьба. То есть, в зависимости от состояния памяти компьютера, компилятор сам решает, делать ли функцию `inline`, или нет. Никогда не будет `inline`-функцией, метод, содержащий в своем теле любой из циклов, или метод с очень большим телом.

Для закрепления прочитанного материала реализуем методы класса `Point`:

```
#include <iostream.h> // Подключение библиотеки объектного
// ввода-вывода
class Point{
private: // скрытые (внутренние) поля класса
    int x;
    int y;
public:
    Point (int x1,int y1); // Первый конструктор
    Point();// Второй конструктор
    void Show();// Метод вывода на печать
    int GetX() // Реализация метода, возвращающего значение x
    {return x;}
    int GetY()// Реализация метода, возвращающего значение y
    {return y;}
    void PutX(int X);// Метод, устанавливающий новое значение
    void PutY(int Y);
};
Point :: Point(int x1, int y1){
x = x1;
y = y1;
}
Point :: Point(){
x = 0;
y = 0;
}
void Point:: Show(){
cout << "Точка (" << x << ", " << y << ")" << endl;
```

```

}
void Point::PutX(int X){
x = X;
}
void Point::PutY(int Y){
y = Y;
}

```

Порядок выполнения работы

1. Получить индивидуальный вариант
2. Доработать описание класса из предыдущей лабораторной работы
3. Написать программу, реализующую решение задания индивидуального варианта
4. Отладить и протестировать программу
5. Защитить программу

Контрольные вопросы

1. Сколько методов в разработанном Вами классе?
2. Как можно написать реализацию методов в классе?
3. Какой метод никогда не будет `inline`?
4. Могут ли присутствовать в классе методы с одинаковыми именами?

1.13 Лабораторная работа «Конструкторы и деструкторы»

Цель работы – получить навыки проектирования и разработки объектно-ориентированных программ.

Форма проведения – выполнение индивидуального задания.

Рекомендации по выполнению работы

Конструктор – это метод класса, который обязательно носит такое же имя, как и класс, и не имеет возвращаемого значения. Конструктор вызывается при описании объекта класса, либо при выделении памяти под объект и отвечает за распределение памяти для хранения объекта класса. Если необходимо проинициализировать поля класса какими-либо начальными значениями, то это, как правило, выполняется в конструкторе. Класс может иметь несколько конструкторов, которые обязательно должны различаться списками параметров. Конструкторы могут быть описаны любыми атрибутами доступа. Естественно, конструктор, имеющий атрибут доступа `private`, может использоваться только в методах класса.

Класс может не иметь конструкторов. В этой ситуации средствами C++ создается конструктор по умолчанию. Этот конструктор не имеет параметров и отвечает только за распределение памяти, необходимой для хранения объекта.

Рассмотрим примеры классов:

Обратите внимание – память под поля-указатели выделяется в конструкторе.

```

class Student{
char * name;
int age;
char *address;
// закрытый конструктор, устанавливающий нулевые
// значения для полей объекта
Student(){
name = NULL;

```



```

address = NULL;
age = 0;
}
public:
// Конструктор, который выделяет память и инициализирует
// значения полей класса
Student(char * name1, int age1, char * address)
{
name = new char [strlen(name1)+1];
strcpy(name, name1);
int age = age1;
address = new char [strlen(address1)+1];
strcpy(address, address1);
}
// Конструктор, запрашивающий поля объекта с клавиатуры
Student(int fl)
{
char name1[100];
cout << "Введите имя студента: ";
cin.getline(name1);
name = new char [strlen(name1)+1];
strcpy(name, name1);
cout << "Введите возраст: ";
cin >> age;
char address1[100];
cout << "Введите домашний адрес: ";
cin.getline(address1);
address = new char [strlen(address1)+1];
strcpy(address, address1);
}
...
// метод, использующий закрытый конструктор:
void Met1(){
...
Student S1;
...
}
}
// Класс Pixel без конструктора
class Pixel{
int x,y;
public:
void PutX(int x1);
void PutY(int y1);
int GetX(){return x;};
int GetY(){return y;};
};
void Pixel::PutX(int x1){
x = x1;
}

```

```
void Pixel::PutY(int y1){
    y = y1;
}
```

При описании объекта класса `Pixel` будет вызываться конструктор по умолчанию. В Си++ существует специальный вид конструктора – конструктор копирования. Задача конструктора копирования – создать объект, являющийся копией уже имеющегося объекта.

Синтаксис конструктора копирования:

```
<Имя класса>(<Имя класса>& <имя переменной>);
```

Например, описание конструктора копирования для класса `Pixel` будет выглядеть следующим образом:

```
Pixel(Pixel &a);
```

Реализация конструктора будет следующей:

```
Pixel::Pixel(Pixel &a)
{
    x = a.x;
    y = a.y;
}
```

Параметром конструктора копирования указывается ссылка на объект, копию которого Вы хотите создать. Если в классе есть поля, то при создании копии эти поля обязательно должны быть определены. В рассмотренном примере поля `x` и `y` задаются значениями полей `x` и `y` объекта `a`.

Вызов конструктора копирования продемонстрируем в следующем примере:

```
...
Pixel n; // описание объекта n
// инициализация полей объекта с помощью методов
n.PutY(10);
n.PutX(4);
// Создание копии объекта
Pixel m(n);
// Обратите внимание – параметром конструктору
// копирования передается объект n, а не ссылка на него.
// Имена объектов (так же как и массивов) являются
// и ссылками на объект
...

```

После работы этого фрагмента в объекте `m` поля `x` и `y` равны соответственно 4 и 10.

Деструктор – это специальный метод, который предназначен для освобождения памяти из под объекта. Деструктор называется по имени класса с лидирующим знаком `~` (тильда) и не имеет параметров. В классе может быть только один деструктор. Если деструктор в классе не описан, то он создается средствами языка Си. Как правило, деструкторы пишутся в том случае, если для полей объекта выделялась память. В этом случае в деструкторе ее освобождают. Для класса `Student` можно описать деструктор следующим образом:

```
~Student();
Реализация:
Student::~~Student(){
    delete [] name;
    delete [] address;
}
```

Деструктор вызывается, если объект заканчивает свое существование или если происходит освобождение памяти, занятой объектом. Продемонстрируем механизм вызова деструктора на следующем примере:

```
//Допишем в класс Pixel деструктор
~Pixel(){
cout << "By!!!";
}
// Напишем стартовый класс:
class St{
public:
void Run();
};
void St::Run(){
Pixel my;
Pixel *my1;
my1 = new Pixel();
my.PutX(12);
my.PutY(13);
my1->PutX(10);
my1->PutY(12);
if (my1->GetX() < 25) delete my1;// при выполнении этой
// строки программы на экране выведется «By!!!» -
// закончит существование объект my1
cout << my.GetX();
} // при окончании работы программы закончит свое
// существование объект my, автоматически вызовется его
// деструктор, на экране еще раз выведется строка «By!!!»
```

Порядок выполнения работы

1. Получить индивидуальный вариант
2. Доработать описание класса из предыдущей лабораторной работы
3. Написать программу, реализующую решение задания индивидуального варианта
4. Отладить и протестировать программу
5. Защитить программу

Контрольные вопросы

1. Дайте определение конструктора.
2. Каково назначение конструктора?
3. Перечислите отличия конструктора от метода.
4. Сколько конструкторов может быть в классе?
5. Допускается ли перегрузка конструкторов?
6. Какие виды конструкторов создаются по умолчанию?
7. Может ли конструктор быть приватным?
8. Какие последствия влечет за собой объявление конструктора приватным?
9. Что такое деструктор?
10. Может ли деструктор иметь параметры?
11. Допускается ли перегрузка деструкторов?

1.14 Лабораторная работа «Массивы объектов»

Цель работы – научиться создавать массивы объектов.

Форма проведения – выполнение индивидуального задания.

Рекомендации по выполнению работы. В программах на языке C++ можно создать 2 разных вида массивов объектов.

Статические массивы объектов

Синтаксис описания:

<Имя класса> <Имя массива> [Размер массива]

Такое описание может быть выполнено только при наличии в классе конструктора без параметров. Если конструктор без параметров в классе не определен, то в программе можно описать статический массив ссылок на объекты.

Синтаксис описания:

<Имя класса> *<Имя массива> [Размер массива]

В этом случае необходимо выделить память под все объекты массива. На рисунке 1.14 продемонстрировано описание статического массива объектов.

```
main.cpp
1  #include <iostream>
2  #include <cstdlib>
3  using namespace std;
4  /* run this program using the console pauser or ad
5  class Pixel{
6      private:
7          int x,y;
8      public:
9          Pixel(){
10         };
11         Pixel(int x,int y) {
12             this->x = x;
13             this->y = y;
14         }
15         void Show(){
16             cout << "Pixel (" << x << ", "
17                 << y << ")" << endl;
18         }
19         void PutX(int x){
20             this->x = x;
21         }
22         void PutY(int y){
23             this->y = y;
24         }
25     };
26 int main(int argc, char** argv) {
27     Pixel A[5];
28     for(int i=0;i<5;i++){
29         A[i].PutX(rand()%20);
30         A[i].PutY(rand()%20);
31     }
32     for(int i=0;i<5;i++)
33         A[i].Show();
34     return 0;
35 }
```

Рисунок 1.14 – Реализация статического массива объектов класса Pixel

В примере создан массив объектов класса Pixel из пяти элементов. Поля x и y для каждого из пяти объектов не заданы, так как для создания объектов использовался конструктор без параметров.

Поля объектов инициализируются отдельно, вызовами методов PutX и PutY. На рисунке 1.15 представлен скриншот, демонстрирующий вызов функции main.

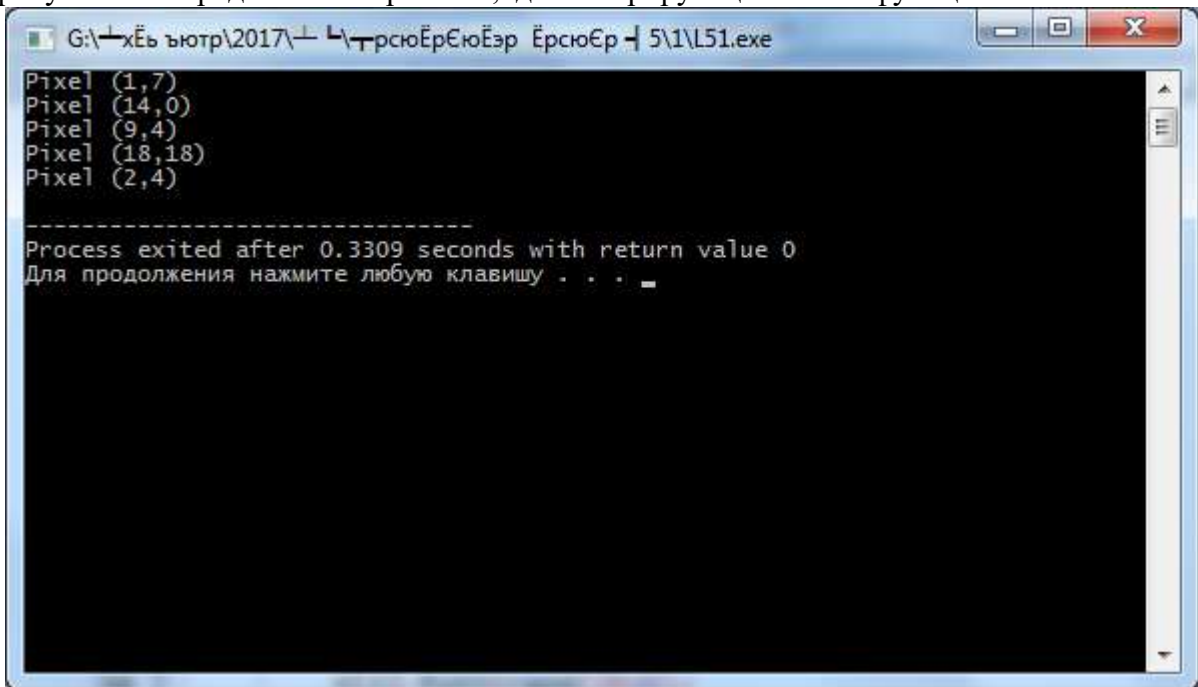


Рисунок 1.15 – Результат создания статического массива объектов

Добавим в описание функции main описание массива из ссылок на объекты так, как это показано на рисунке 1.16.

```
Pixel * B[5];  
cout << "Array B" << endl;  
for (int i=0;i<5;i++) {  
    B[i] = new Pixel(rand()%20,rand()%20);  
    B[i]->Show();  
}
```

Рисунок 1.16 – Описание статического массива ссылок на объекты

Для каждой ссылки на объект выделена память с помощью конструктора с параметрами. Обратите внимание, элементы массива в этом случае – ссылки. На рисунке 1.17 представлен результат работы программы:

```

Pixel (1,7)
Pixel (14,0)
Pixel (9,4)
Pixel (18,18)
Pixel (2,4)
Array B
Pixel (5,5)
Pixel (1,7)
Pixel (1,11)
Pixel (15,2)
Pixel (7,16)
-----
Process exited after 0.3084 seconds with return value 0
Для продолжения нажмите любую клавишу . . .

```

Рисунок 1.17 – Результат создания статического массива динамических объектов

Динамические массивы объектов

Динамические массивы объектов могут быть организованы двумя способами: динамические массивы объектов и динамические массивы ссылок на объекты. Динамические массивы объектов можно объявить в программе следующим образом:

<Имя класса> * <Имя массива> ;

Выделение памяти под массив:

<Имя массива> = new <Имя класса>[количество элементов массива];

Такой синтаксис выделения памяти может быть использован только в случае наличия в классе конструктора без параметров.

На рисунке 1.18 показан пример описания такого массива:

```

Pixel *C;
int n;
cout << "Enter n: ";
cin >> n;
C = new Pixel[n];
for(int i=0;i<n;i++){
    C[i].PutX(rand()%20);
    C[i].PutY(rand()%20);
    cout << "Array C" << endl;
}
for (int i=0;i<n;i++) {
    C[i].Show();
}

```

Рисунок 1.18 – Динамический массив статических объектов

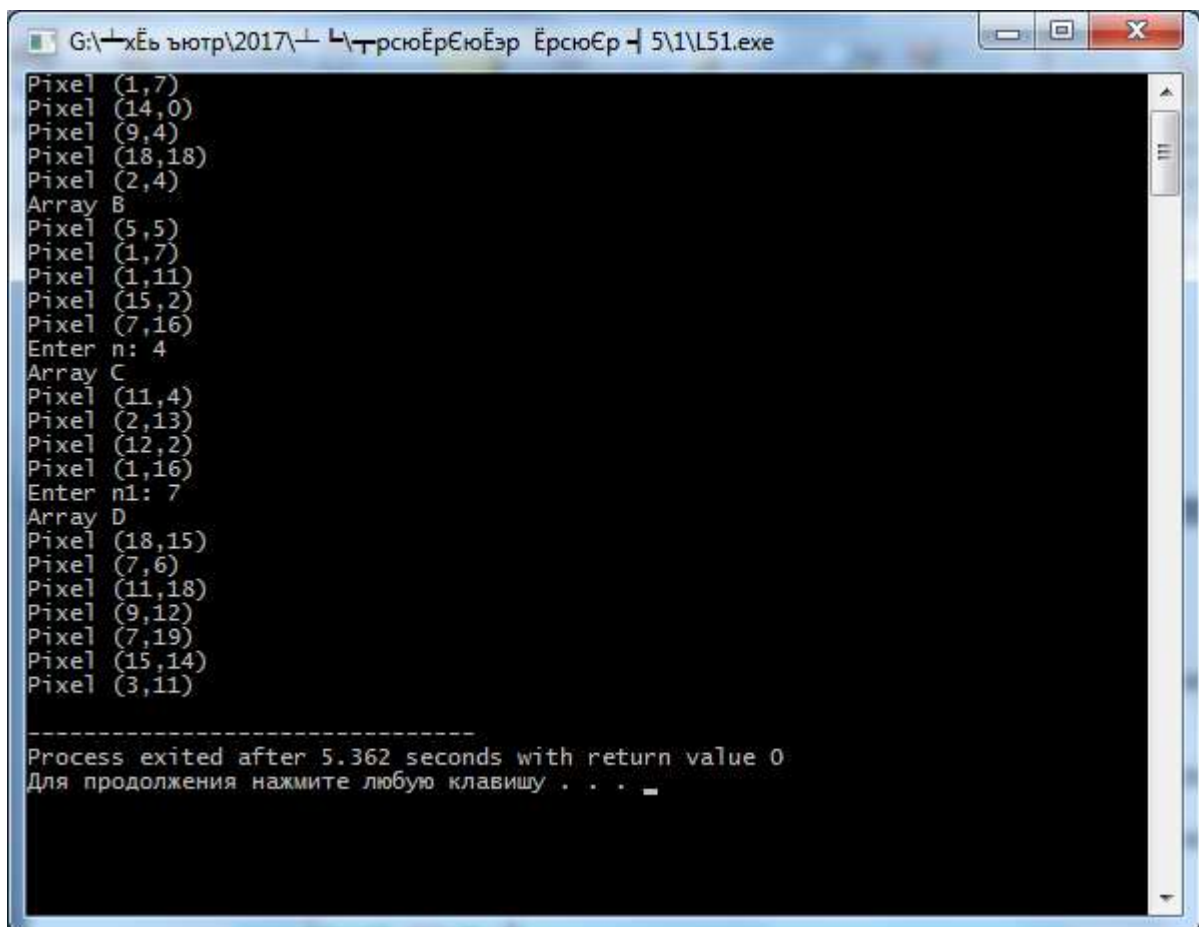
Размер массива вводится с клавиатуры, поля объектов формируются вызовами методов PutX, PutY.

Последний способ описания массивов объектов – динамический массив ссылок на объекты. В этом случае необходимо выделить память отдельно для массива ссылок и для каждого элемента массива, так как это показано в примере на рисунке 1.19.

```
Pixel ** D; |
int n1;
cout << "Enter n1: ";
cin >> n1;
D = new Pixel*[n1];
for(int i=0;i<n1;i++){
    D[i] = new Pixel(rand()%20,rand()%20);
}
cout << "Array D" << endl;
for (int i=0;i<n1;i++) {
    D[i]->Show();
}
```

Рисунок 1.19 – Динамический массив динамических объектов

Результат работы программы представлен на рисунке 1.20.



```
G:\ХЕЬ ЁЮТР\2017\... 5\1\L51.exe
Pixel (1,7)
Pixel (14,0)
Pixel (9,4)
Pixel (18,18)
Pixel (2,4)
Array B
Pixel (5,5)
Pixel (1,7)
Pixel (1,11)
Pixel (15,2)
Pixel (7,16)
Enter n: 4
Array C
Pixel (11,4)
Pixel (2,13)
Pixel (12,2)
Pixel (1,16)
Enter n1: 7
Array D
Pixel (18,15)
Pixel (7,6)
Pixel (11,18)
Pixel (9,12)
Pixel (7,19)
Pixel (15,14)
Pixel (3,11)
-----
Process exited after 5.362 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1.20 – Создание динамического массива динамических объектов

Освобождение памяти происходит в порядке, обратном выделению памяти (для всех объектов программы, которые были сформированы с помощью оператора new). На

рисунке 1.21 показана реализация освобождения памяти при использовании разных типов массивов объектов.

```
for (int i=0;i<5;i++)
    delete B[i];
delete [] C;
for (int i=0;i<n1;i++)
    delete D[i];
delete [] D;
```

Рисунок 1.21 – Освобождение памяти

Для массива В – освобождение памяти для каждого объекта (сам массив – статический, память не освобождается).

Для массива С – освобождается память только из-под массива (сами объекты – статические).

Для массива D – память освобождается сначала из-под объектов, затем из под самого массива.

Порядок выполнения работы

1. Получить индивидуальный вариант
2. Реализовать создание массива объектов
3. Написать программу, реализующую решение задания индивидуального варианта
4. Отладить и протестировать программу
5. Защитить программу

Контрольные вопросы

1. Можно ли создавать массивы объектов?
2. Сколько видов массивов можно создать?
3. Как создание массивов связано с конструктором класса?
4. Нужно ли освобождать память при работе с массивами объектов?
5. Как объявить массив объектов класса, если конструктор класса имеет параметры?

1.15 Лабораторная работа «Перегрузка операторов»

Цель работы – реализовать механизм перегрузки операторов для объектов написанного класса.

Форма проведения – выполнение индивидуального задания.

Рекомендации по выполнению работы

Для наглядности текстов программ, написанных в стиле ООП, в классах можно переопределить операторы. Пусть, например, описан класс для работы с матрицами *Matrix*. В классе описан метод *Mult(Matrix a)*, получающий произведение двух матриц. Если в программе описаны два объекта класса *Matrix m1* и *m2*, то их произведение можно получить следующим образом: *Matrix c = m1.Mult(m2);*

Если бы в классе были перегружены операторы ‘*’ и ‘=’, то запись этого же действия выглядела бы следующим образом: *Matrix c = m1*m2;*

Очевидно, что вторая запись более привычна. Синтаксис перегрузки бинарных операторов:

<возвращаемый тип> *operator* <оператор>(тип второго операнда)


```
{ тело перегруженного оператора
}
```

Синтаксис перегрузки унарных операторов:

```
<возвращаемый тип> operator <оператор>()
```

```
{ тело перегруженного оператора
}
```

Внимание! Первый операнд всегда – объект класса, которому принадлежит перегруженный оператор. Стандартными перегруженными операторами являются операторы '<<' и '>>'.

В описанном ниже примере рассматривается перегрузка операторов сложения и присваивания:

```
class Matrix{
private: int n,m;
        int **x;
public:
Matrix();
Matrix(int n1,int m1);
Matrix& Create(int n1, int m1);
void Print(char * string);
void Put(int i, int j, int x1);
int Get(int i, int j);
Matrix& operator * (Matrix &a);
Matrix& operator = (Matrix &a);
~Matrix();
};
// конструктор без параметров, этот конструктор
// инициализирует значения полей класса нулевыми
// значениями
Matrix::Matrix(){
n = 0;
m = 0;
x = NULL;
}
// конструктор, инициализирующий количество строк и
// столбцов матрицы заданными значениями; а так же,
// этот конструктор выделяет память под матрицу с уже
// известными размерами и инициализирует значения
// элементов матрицы случайными числами
Matrix::Matrix(int n1, int m1){
n = n1;
m = m1;
x = new int* [n];
for (int i=0;i<n;i++)
    x[i] = new int [m];
for(i=0;i<n;i++)
    for(int j=0;j<m;j++)
        x[i][j] = random(5);
}
// метод, выводящий матрицу на экран с заданным
// заголовком string
```

```

void Matrix::Print(char* string){
if (x == NULL)
cout << endl << "Матрица объекта не задана."<< endl;
else{
cout << string << endl;
for(int i=0;i<n;i++)
{ for(int j=0;j<m;j++)
cout << x[i][j] << " ";
cout << endl;
}
}
}
// метод, изменяющий значение элемента,
// расположенного в позиции i,j на значение x1
inline void Matrix :: Put(int i, int j, int x1){
x[i][j] = x1;
}

// метод, возвращающий значение элемента матрицы,
// расположенного в позиции i,j
inline int Matrix :: Get(int i, int j){
return x[i][j];
}
// перегруженный оператор умножения двух матриц
Matrix& Matrix :: operator * (Matrix &a){
// указатель на объект
Matrix *temp;
// выделение памяти под пока пустую матрицу
temp = new Matrix();
// проверка возможности выполнить операцию умножения
if (m!=a.n) {
cout << endl << "Операция не может быть выполнена." << endl;
return *temp;}
// выделение памяти под матрицу – результат умножения
temp->Create(n,a.m);
// умножение матриц
for (int i=0;i<n;i++)
for(int j=0;j<a.m;j++)
{ int s = 0;
for (int k=0;k<m;k++)
s+= x[i][k]*a.x[k][j];
temp->x[i][j] = s;
}
// возвращение результата
return *temp;
}
// перегруженный оператор присваивания,
// обратите внимание – при использовании этого
// оператора должен получиться объект, равный

```

```

// заданному, поэтому в теле оператора определяются
// все поля объекта, подумайте, что произойдет, если
// не определить какие-либо поля
Matrix& Matrix :: operator = (Matrix& a){
    n = a.n;
    m = a.m;
    if (x!=NULL) {
        for(int i=0;i<n;i++)
            delete [] x[i];
        delete [] x; }
    x = new int* [n];
    for(int i=0;i<n;i++)
        x[i] = new int [m];
    for(i=0;i<n;i++)
        for(int j=0;j<m;j++)
            x[i][j] = a.x[i][j];
    return *this;
}
// метод, выделяющий память под матрицу объекта и
// задающий новые значения для полей n,m; элементы
// матрицы при этом не инициализируются
Matrix& Matrix :: Create(int n1, int m1){
    n = n1;
    m = m1;
    if (x!=NULL) {
        for(int i=0;i<n;i++)
            delete [] x[i];
        delete [] x; }
    x = new int* [n];
    for(int i=0;i<n;i++)
        x[i] = new int [m];
    return *this;
}
// деструктор класса
Matrix::~Matrix(){
    for(int i=0;i<n;i++)
        delete [] x[i];
    delete [] x;
}
// пример использования перегруженных операторов
void main(){
    randomize();
    // описание и создание объектов A (матрица размерности
    // 3x4 ) и B (матрица размерности 4x5 )
    Matrix A(3,4);
    Matrix B(4,5);
    // печать исходных матриц
    A.Print("Первая матрица:");
    B.Print("Вторая матрица:");
    // получение матрицы C – результат умножения матриц

```

```

// A и B
  Matrix C = A*B;
// печать матрицы C
  C.Print("A*B = ");
}

```

Операции `<<`, `>>` являются перегруженными в классах `istream` (входные потоки) и `ostream` (выходные), наследуются всеми производными от них классами. Для собственного класса можно перегрузить операцию вставки для вывода отдельных или всех полей класса, однако, необходимо пользоваться определенными правилами. Перегрузим операцию вставки для класса `VECTOR`. В описание класса добавим:

```

friend ostream& operator << (ostream& os, VECTOR& a);
// ostream& os – ссылка на класс-хозяин
Определим дружественную функцию:
ostream& operator << (ostream& os,VECTOR& a)
{
  os << "-----" << endl;
  os << "Size vector : " << a.n <<endl;
  os << "Elements vector : " << endl;
  for (int i = 0; i< a.n; i++)
    {
      os << "vector[" << i << "]" = " << a.vector[i] << endl;
    }
  os << "-----" << endl;
  return os;
}

```

После этого в программе может быть следующее выражение:

```

VECTOR a(10);
cout << a;

```

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Реализовать перегруженные операторы для разработанного ранее класса.
3. Написать программу, реализующую решение задания индивидуального варианта.
4. Отладить и протестировать программу.
5. Защитить программу.

Контрольные вопросы

1. Для чего используются перегруженные операторы?
2. Приведите пример перегруженных операторов.
3. Какое ключевое слово используется при реализации перегруженных операторов?
4. Почему некоторые перегруженные операторы лучше реализовать как дружественные функции?
5. Какие операторы разрешает перегружать стандарт C++?

1.16 Лабораторная работа «Наследование. Создание базового класса и класса-наследника»

Цель работы – получить навыки проектирования иерархии классов.

Форма проведения – выполнение индивидуального задания.

Рекомендации по выполнению работы.

Базовые и производные классы. Наследование – это способность брать существующий (базовый класс) и порождать из него новый класс – потомок, с наследованием всех его атрибутов и поведения. Способность к наследованию, возможно, является, единственным коренным отличием C++ от Си. **Классы – потомки** будем называть производными классами.

Синтаксис определения производного класса:

```
class <имя производного класса> : public <имя базового
класса> { описание полей и методов класса
}
```

Опишем класс для работы с графической точкой:

```
class pixel
{ protected:
  int x,y,color;// значения координат и цвет
public:
  pixel(int a,int b,short c); // Конструктор
  void change(int dx, int dy);// Изменение координат
  void draw(const int = -1);// Рисование точки
  int getX();// Получение координаты x
  int getY();// Получение координаты y
  int getColor();// Получение цвета
};
```

Опишем производный класс `circle_` (окружность). Поля `x,y,color` наследуются производным классом – у окружности есть координаты центра и на экран окружность будет выводиться заданным цветом. Но окружность характеризуется еще и радиусом – поэтому в производном классе появляется еще одно поле – `rad`. Так как конструкторы не наследуются, то в классе `circle_` опишем конструктор, который задает поля класса `x,y,color` и `rad`. Метод изменения координат наследуется производным классом. Метод рисования переопределяется в производном классе, поскольку рисование окружности отличается от рисования точки. Методы `getX()`, `getY()` и `getColor()` наследуются производным классом, в класс добавляется новый метод `getR()`, который будет возвращать значение поля `rad`.

```
class circle_ : public pixel
{ protected:
  int rad;// радиус окружности
public:
  circle_(int a,int b,short c, int r); // Конструктор
  void draw(const int = -1);// Рисование окружности
  int getR();// Получение радиуса
};
```

Обратите внимание: наследуемые поля и методы не описываются в производных классах. По определению наследования эти элементы уже существуют в классе.

Ниже приведена реализация методов классов `pixel` и `circle_`.

```
// конструктор
pixel :: pixel (int a, int b, short c)
```

```

{
x = a;
y = b;
color = c;
}
// изменение координат
void pixel :: change(int dx, int dy)
{
x+=dx;
y+=dy;
}
// рисование точки; параметр c, по умолчанию равный -1,
// необходим для того, чтобы использовать функцию
// рисования и для того, чтобы стирать нарисованную точку
void pixel :: draw(const int c)
{ // если функция вызывается без параметров
if (c== -1)
    putpixel(x,y,color); // то рисовать точку цветом color
    else putpixel(x,y,c); // иначе - рисовать точку цветом,
// заданным в параметрах функции }
int pixel :: getX(){
return x;}
int pixel :: getY(){
return y; }
int pixel :: getColor() {
return color; }
// реализация методов класса circle_
// обратите внимание на синтаксис написания конструктора
// через знак «:» вызывается конструктор базового класса
// при его вызове инициализируются поля x,y,color;
// следовательно, в самом конструкторе нужно
// проинициализировать только поле rad
circle_ :: circle_(int a, int b, short c, int r) : pixel(a,b,c)
{
    rad = r; }
int circle_ :: getR() {
return rad; }
// переопределяемый метод рисования окружности,
// принцип работы функции такой же, как и в базовом
// классе - анализируется значение параметра c, в
// зависимости от этого устанавливается цвет рисования
void circle_ :: draw(const int c)
{
if (c== -1){
setcolor(color);
circle(x,y,rad);
}
else {setcolor(c);
circle(x,y,rad);
}
}

```

```

// Демонстрация работы с объектами описанных классов:
class MyGraph
{
    public:
    void Run(); };
void MyGraph :: Run () {
    int gdriver = DETECT, gmode, errorcode;
    // проинициализировать графику
    initgraph(&gdriver, &gmode, "d:\lang\bc\bgi");
    // описать объект «точка» - координаты 200, 200, цвет -
    // красный
    pixel p1(200,200,4);
    // описать объект «окружность» - координаты центра
    //200, 200, радиус - 45, цвет - зеленый
    circle c(200,200,2,45);
    // нарисовать объекты
    p1.draw();
    c.draw();
    // ожидание нажатия клавиши
    getch();
    // стереть объекты, используя для рисования
    // полученный текущий цвет фона
    p1.draw(getbkcolor());
    c.draw(getbkcolor());
    closegraph();
}
// Функция main:
void main()
{
    MyGraph obj;
    obj.Run();
}

```

В C++ предусмотрен удобный механизм выделения памяти для объектов, находящихся в одной иерархии наследования. Если в программе описан указатель на объект базового класса, то указывать в дальнейшем он может и на объекты производных классов, но не наоборот. То есть в программе могут быть следующие строки:

```

pixel *obj;
int k;
cout << "Введите 1, для работы с объектом базового
класса или 2, для работы с объектом производного класса";
cin >> k;
if(k==1) obj = new pixel(40,40,6);
else obj = new circle_(40,40,2,12);

```

Полиморфизм. Статический полиморфизм – это использование переопределяемых функций. В предыдущем примере еще на этапе компиляции объект `c` был связан с функцией `draw` для класса `circle_`, а объект `p1` – с функцией `draw` для класса `pixel`. Такой полиморфизм называют еще механизмом раннего связывания.

Динамический полиморфизм. Допишем к описаниям наших классов класс `action`, который будет заставлять двигаться наши объекты.

```

class action

```

```

{
  public :
  void draw(pixel *a); //
  void clear(pixel *a);
  void move(pixel *a,int dX, int dY);
};
void action::draw(pixel *a) {
  a->draw(); }
void action::clear(pixel *a) {
  int color = getbkcolor();
  a->draw(color); }
void action::move(pixel *a,int dx,int dy) {
  this -> clear(a);
  a->change(dx,dy);
  this -> draw(a); }

```

По правилам определения объектов одной иерархии наследования класс `action` может работать и с объектами класса `pixel` и с объектами класса `circle_`. Но при следующей реализации метода `Run` программа будет работать некорректно:

```

class MyGraph
{
  public:
  void Run(); };
void MyGraph :: Run () {
  int gdriver = DETECT, gmode, errorcode;
  // проинициализировать графику
  initgraph(&gdriver, &gmode, "d:\lang\bc\bgi");
  action b;
  pixel * obj;
  int k;
  cout << "Введите 1, для работы с объектом базового
  класса, 2 - для работы с объектом производного класса: ";
  cin >>k;
  if (k==1)obj = new pixel(100,100,14);
  else obj = new circle_(100,100,5,45);
  obj->draw();
  for(int i=0;i<15;i++)
  { b.move(obj,15,15);
    delay(300);
  }
  // ожидание нажатия клавиши
  getch();
  closegraph();
}
// Функция main:
void main()
{
  MyGraph obj;
  obj.Run();
}

```


При всех значениях переменной `k` двигаться по экрану будет объект «точка». В данном примере работает механизм раннего связывания и не важно, каким образом выделена память объект `obj` всегда будет обращаться к методу `draw` класса `pixel`.

Для решения этой проблемы в C++ предусмотрен так называемый механизм позднего связывания: переопределяемые функции в описании класса объявляют виртуальными (ключевое слово `virtual`). При компиляции таких классов в памяти компьютера создается таблица виртуальных методов `vtbl`. В этой таблице хранятся адреса всех виртуальных методов. При компиляции программы каждый объект дополняется ссылкой на `vtbl` – `vptr`, значение этой ссылки сам компилятор и заполняет. На этапе компиляции все вызовы виртуальных методов заменяются на значение `vptr`, а на этапе выполнения выбирается адрес из `vtbl`.

Рекомендуется делать виртуальными деструкторы для того, чтобы гарантировать правильное освобождение памяти из под объекта. Так как при этом будет вызван именно тот деструктор, который нужен. Таким образом для того, чтобы описанная выше программа работала корректно, необходимо в описаниях классов `pixel` и `circle_` сделать виртуальными методы `draw`.

Порядок выполнения работы

1. Получить индивидуальный вариант.
2. Спроектировать иерархию классов «базовый класс – класс наследник».
3. Написать программу, реализующую решение задания индивидуального варианта и демонстрирующую принцип динамического полиморфизма.
4. Отладить и протестировать программу.
5. Защитить программу.

Контрольные вопросы

1. Дайте определение наследования.
2. Какие бывают типы наследования?
3. Существует ли в Си++ множественное наследование?
4. Дайте определение статического полиморфизма.
5. Дайте определение динамического полиморфизма.
6. Что происходит в памяти компьютера при описании в классе хотя бы одной виртуальной функции?

1.17 Лабораторная работа «Работа с потоковыми классами C++»

Цель работы – получить навыки проектирования и разработки объектно-ориентированных программ.

Форма проведения – выполнение индивидуального задания.

Рекомендации по выполнению работы.

Функционал потоков ввода/вывода не определен как часть языка C++, а предоставляется Стандартной библиотекой C++ (и, следовательно, находится в пространстве имен `std`). На предыдущих занятиях вы подключали заголовочный файл библиотеки `iostream` и использовали объекты `cin` и `cout` для простого ввода/вывода данных. На этой лабораторной работе будет детально рассмотрена библиотека `iostream`.

При подключении заголовочного файла `iostream`, получен доступ ко всей иерархии классов библиотеки `iostream`, отвечающих за функционал ввода/вывода данных (включая класс, который называется `iostream`).

Первое, что можно заметить в этой иерархии – множественное наследование. Тем не менее, библиотека `iostream` была разработана и тщательно протестирована соответствующим образом, дабы избежать типичных ошибок, которые возникают при работе с множественным наследованием.

Второе, что можно было заметить – это частое использование слова «`stream`». По сути, ввод/вывод в языке C++ реализован с помощью потоков. Абстрактно, поток – это последовательность символов, к которым можно получить доступ. Со временем поток может производить или потреблять потенциально неограниченные объемы данных.

Рассмотри два типа потоков. Поток ввода (или «входной поток») используется для хранения данных, полученных от источника данных: клавиатуры, файла, сети. Например, пользователь может нажать клавишу на клавиатуре в то время, когда программа не ожидает ввода. Вместо игнорирования нажатия клавиши, данные помещаются во входной поток, где затем ожидают ответа от программы.

И наоборот, поток вывода (или «выходной поток») используется для хранения данных, предоставляемых конкретному потребителю данных: монитору, файлу, принтеру и другим. При записи данных на устройство вывода это устройство может быть не готовым принять данные немедленно. Таким образом, данные будут находиться в потоке вывода до тех пор, пока поток не начнет их использовать.

Некоторые устройства, такие как файлы и сети, могут быть источниками как ввода, так и вывода данных.

Библиотека `iostream` организована таким образом, что программисту не нужно знать детали взаимодействия потоков с разными устройствами и источниками данных, ему нужно только научиться взаимодействовать с этими потоками для чтения и записи данных.

Класс `ios` определяет множество элементов, которые являются общими для потоков ввода/вывода.

Класс `istream` используется для работы с входными потоками. Оператор извлечения `>>` используется для извлечения значений из потока. Когда пользователь нажимает на клавишу клавиатуры, код этой клавиши помещается во входной поток. Затем программа извлекает это значение из потока и использует его.

Класс `ostream` используется для работы с выходными потоками. Оператор вставки `<<` используется для помещения значений в поток. Пользователь вставляет свои значения в поток, а затем потребитель данных (например, монитор) использует их.

Класс `iostream` может обрабатывать как ввод, так и вывод данных, что позволяет ему осуществлять двунаправленный ввод/вывод.

Стандартный поток – это предварительно подключенный поток, который предоставляется программе её окружением. Язык C++ поставляется с четырьмя предварительно определенными стандартными объектами потоков, которые можно использовать:

`cin` – класс `istream_withassign`, связанный со стандартным вводом (обычно это клавиатура);

`cout` – класс `ostream_withassign`, связанный со стандартным выводом (обычно это монитор);

`cerr` – класс `ostream_withassign`, связанный со стандартной ошибкой (обычно это монитор), обеспечивающий небуферизованный вывод;

`clog` – класс `ostream_withassign`, связанный со стандартной ошибкой (обычно это монитор), обеспечивающий буферизованный вывод.

Порядок выполнения работы

1. Получить индивидуальный вариант.

2. Спроектировать алгоритм, указанный в индивидуальном задании с использованием потоковых классов языка C++.
3. Написать программу, реализующую решение задания индивидуального варианта.
4. Отладить и протестировать программу.
5. Защитить программу.

Контрольные вопросы

1. Что такое «поток»?
2. Как классифицируются потоки, реализованные в библиотеках ввода/вывода C++?
3. Что такое буферизация и зачем она нужна?
4. Какие библиотеки ввода/вывода реализованы в C++ и чем они отличаются?
5. Перечислите стандартные потоки и объясните их назначение.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

2.1 Общие положения

Самостоятельная работа является важной составляющей в изучении дисциплины и состоит из следующих видов деятельности: проработка лекционного материала для подготовки к тестированию и контрольным работам, подготовка к лабораторным работам, выполнение домашних заданий, выполнение контрольных работ, самостоятельное изучение тем курса.

Самостоятельная работа над теоретическим материалом направлена на систематизацию и закрепление знаний, полученных на лекционных занятиях и на получение новых знаний по дисциплине, путем самостоятельного изучения тем.

Самостоятельная работа по подготовке к лабораторным работам направлена на изучение методического и теоретического материала по теме лабораторной работы.

Выполнение домашних заданий и контрольных работ – полностью самостоятельная работа, направленная на получение навыков самостоятельного составления алгоритмов, реализацию программ, их дальнейшей отладки и тестирования.

2.2 Проработка лекционного материала, подготовка к контрольным работам, лабораторным работам и практическим занятиям

Проработка лекционного курса является одной из важных активных форм самостоятельной работы. Этот вид самостоятельной работы может быть организован следующим образом:

- прочитайте конспект лекции, согласовав Ваши записи с информацией на слайдах лекции;
- попробуйте выполнить самостоятельно примеры программ, разобранных на лекции;
- если в лекции рассматривался какой-либо алгоритм, попытайтесь выполнить этот алгоритм на тестовых данных без использования компьютерной программы; такой способ проработки материалов лекции покажет, правильно ли Вы поняли идею алгоритма;
- изучите дополнительные учебные материалы, рекомендованные преподавателем;
- попытайтесь ответить на контрольные вопросы, которыми, как правило, заканчиваются разделы учебных пособий или учебников;
- если после выполненной работы Вы считаете, что материал освоен не полностью, сформулируйте вопросы и задайте их преподавателю.

Методические указания к ведению конспектов лекций. Лекции по дисциплине проводятся с использованием слайдов. Но это не означает, что лекцию можно просто слушать. Ведение конспектов значительно повышает качество последующей проработки лекционного материала. В силу специфики дисциплины на слайдах лекций очень много алгоритмов, кодов программ, примеров демонстрации работы изучаемых алгоритмов. Но этот материал может быть бесполезен, если Вы не делаете записи в течение лекции, потому что в большинстве случаев, комментарии по представленным на слайдах примерам, лектор выполняет в устной форме.

Можно рекомендовать распечатывать слайды перед лекцией и вести конспект непосредственно на бумажном варианте слайд-презентации.

Одной из форм текущего мониторинга уровня знаний по дисциплине являются контрольные работы. В первом и втором семестре изучения дисциплины проводятся контрольные работы двух типов: тестовые опросы на лекции и контрольные работы, в которых студентам необходимо применить полученные знания на практике. Выполнение вышеперечисленных действий поможет подготовиться и к выполнению контрольных работ. В приложении А указаны темы контрольных работ и приведены примерные варианты.

Самостоятельная работа по подготовке к лабораторным работам и практическим занятиям по дисциплине состоит в изучении методических материалов по темам соответствующих видов аудиторных занятий.

Рекомендуется перед выполнением лабораторной работы изучить лекционный и методический материал по теме занятия, ознакомиться с алгоритмами, реализацию которых необходимо выполнить во время проведения занятия. Обратите особое внимание на порядок выполнения работы. Поскольку конечным результатом всех лабораторных работ является компьютерная программа, самостоятельно разработайте структурную схему будущей программы, выполните заготовку проекта, подготовьте самостоятельно тестовые данные. Если при подготовке к занятию остались нерешенные вопросы, обратитесь за консультацией к преподавателю.

2.3 Выполнение домашних заданий и контрольных работ

Выполнение домашних заданий – это полностью самостоятельный вид деятельности студента. Темами домашних заданий и контрольных работ являются темы дисциплины, не охваченные циклом лабораторных работ и практических занятий.

Выполнение домашнего задания состоит в написании программы по индивидуальному варианту. Обучающиеся самостоятельно разрабатывают алгоритм решения задания, реализуют разработанный алгоритм на языке программирования C/C++, отлаживают и тестируют написанную программу.

Защита домашнего задания проходит в сроки, установленные преподавателем. Процедура защиты представляет собой индивидуальное собеседование с преподавателем, примерный сценарий которого представлен ниже:

- комментирование студентом кода и логики написанной программы;
- ответы на вопросы преподавателя по коду и логике программы;
- комментирование студентом тестовых данных;
- демонстрация работы программы и пояснение полученных результатов.

Контрольные работы выполняются самостоятельно, во время семестра. Баллы, полученные за контрольную работу включены в рейтинговую систему.

2.4 Самостоятельное изучение тем теоретической части курса

2.4.1 Файлы в языке Си

Перечень вопросов, подлежащих изучению

1. Описание файловых переменных.
2. Режимы доступа к файлу.
3. Открытие и закрытие файла.
4. Способы чтения информации из файла.

5. Способы записи информации в файл.

Методические рекомендации по изучению

При изучении этой темы следуйте списку вопросов, представленному выше. Для более полного владения темой найдите описание структуры FILE. Запомните существующие режимы доступа к файлу. Обратите внимание, язык программирования Си предоставляет самые разнообразные варианты чтения информации из файла. Отметьте для себя, каким образом можно организовать проверку ошибок чтения или записи в файл. Не забывайте про практическую реализацию приведенных в учебных пособиях примеров. Обратите внимание на многообразие функций языка для организации прямого доступа к файлу. Выделите для себя различия между текстовыми и двоичными файлами. Параллельно с изучением справочного материала попробуйте написать функции записи заданной информации в двоичный файл. Попробуйте открыть двоичный файл в текстовом редакторе. Найдите объяснение увиденной информации.

Рекомендуемые источники

Пермякова, Н. В. Информатика и программирование: Учебное пособие [Электронный ресурс] / Н.В. Пермякова – Томск: ТУСУР, 2016. – 188 с. – Режим доступа: <https://edu.tusur.ru/publications/7678>, стр. 144 – 155.

Подбельский, В.В. Курс программирования на языке Си [Электронный ресурс] : учебник / В.В. Подбельский, С.С. Фомин. – Электрон. дан. – Москва : ДМК Пресс, 2012. – 384 с. – Режим доступа: <https://e.lanbook.com/book/4148> . – Загл. с экрана. Стр. 283 – 321, стр. 322 – 334.

2.4.2 Класс *vector*

Перечень вопросов, подлежащих изучению

1. Описание класса *vector* в библиотеке STL.
2. Свойства класса *vector*.
3. Создание объекта класса *vector*.
4. Изменение объекта класса *vector*.

Методические рекомендации по изучению

При изучении этой темы следуйте списку вопросов, представленному выше. Для более полного владения темой посмотрите реализацию класса. Ознакомьтесь со структурой класса – конструкторы, деструкторы, поля, методы. Обратите внимание на то, что объекты класса реализуют шаблон динамического массива. Отметьте для себя, каким образом можно создать объект класса, как выполнить начальную инициализацию элементов вектора. Не забывайте про практическую реализацию приведенных в учебных пособиях примеров. Обратите внимание на многообразие методов класса. Выделите для себя различия между обращениями к элементам вектора по индексу и по итератору. Параллельно с изучением справочного материала попробуйте написать код, создающий объект класса *vector* и реализующий алгоритмы поиска, сортировки, изменения размера. Попробуйте организовать доступ к элементам вектора через итератор.

Рекомендуемые источники

Ашарина, И. В. Объектно-ориентированное программирование в C++: лекции и упражнения : учебное пособие / И. В. Ашарина. – 2-е изд., перераб. и доп. – Москва : Горячая линия-Телеком, 2017. – 336 с. – ISBN 978-5-9912-0423-1. – Текст : электронный // Лань : электронно-библиотечная система. – URL:

<https://e.lanbook.com/book/119830>. – Режим доступа: для авториз. пользователей. Стр. 297 – 302.

Аммерааль, Л. STL для программистов на C++ / Л. Аммерааль. – Москва : ДМК Пресс, 2006. – 240 с. – ISBN 5-89818-027-3. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/1218>. – Режим доступа: для авториз. пользователей. Стр. 89 - 92, стр. 101 – 102.

Объектно-ориентированное программирование на C++ : учебник / И. В. Баранова, С. Н. Баранов, И. В. Баженова [и др.]. – Красноярск : СФУ, 2019. – 288 с. – ISBN 978-5-7638-4034-6. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/157572>. – Режим доступа: для авториз. пользователей. Стр. 196 – 210.

2.4.3 Класс `string`

Перечень вопросов, подлежащих изучению

1. Описание класса `string` в библиотеке STL.
2. Свойства класса `string`.
3. Создание объекта класса `string`.
4. Изменение объекта класса `string`.

Методические рекомендации по изучению

При изучении этой темы следуйте списку вопросов, представленному выше. Для более полного владения темой изучите реализацию класса. Ознакомьтесь со структурой класса – конструкторы, деструкторы, поля, методы. Обратите внимание на то, как связаны строка `Si` и объект класса `string`. Отметьте для себя, каким образом можно создать объект класса, как выполнить начальную инициализацию элементов строки. Не забывайте про практическую реализацию приведенных в учебных пособиях примеров. Обратите внимание на многообразие методов класса. Выделите для себя различия между строкой `Si` и объектом класса `string`. Параллельно с изучением справочного материала попробуйте написать код, создающий объект класса `string` и реализующий алгоритмы обработки строки.

Рекомендуемые источники

Ашарина, И. В. Объектно-ориентированное программирование в C++: лекции и упражнения : учебное пособие / И. В. Ашарина. – 2-е изд., перераб. и доп. – Москва : Горячая линия-Телеком, 2017. – 336 с. – ISBN 978-5-9912-0423-1. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/119830>. – Режим доступа: для авториз. пользователей. Стр. 309 – 316.

2.4.4 Класс `List`

Перечень вопросов, подлежащих изучению

1. Описание класса `List` в библиотеке STL.
2. Свойства класса `List`.
3. Создание объекта класса `List`.
4. Изменение объекта класса `List`.

Методические рекомендации по изучению

При изучении этой темы следуйте списку вопросов, представленному выше. Для более полного владения темой изучите реализацию класса. Ознакомьтесь со структурой класса – конструкторы, деструкторы, поля, методы. Обратите внимание на то, что объект класса List – реализация динамического однонаправленного списка. Отметьте для себя, каким образом можно создать объект класса, как выполнить начальную инициализацию элементов списка. Не забывайте про практическую реализацию приведенных в учебных пособиях примеров. Обратите внимание на многообразие методов класса. Выделите для себя различия между объектом класса List и объектом класса vector. Параллельно с изучением справочного материала попробуйте написать код, создающий объекты класса List и реализующий алгоритмы обработки элементов списка.

Рекомендуемые источники

Аммерааль, Л. STL для программистов на C++ / Л. Аммерааль. – Москва : ДМК Пресс, 2006. – 240 с. – ISBN 5-89818-027-3. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/1218>. – Режим доступа: для авториз. пользователей. Стр. 211 - 215, стр. 101 – 102.

Объектно-ориентированное программирование на C++ : учебник / И. В. Баранова, С. Н. Баранов, И. В. Баженова [и др.]. – Красноярск : СФУ, 2019. – 288 с. – ISBN 978-5-7638-4034-6. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/157572>. – Режим доступа: для авториз. пользователей. Стр. 20 – 24.

2.4.5 Множественное наследование

Перечень вопросов, подлежащих изучению

1. Синтаксис реализации множественного наследования.
2. Опасные моменты механизма множественного наследования.

Методические рекомендации по изучению

При изучении темы следуйте списку вопросов, представленному выше. Для более полного владения темой представьте реализацию механизма множественного наследования на примере. Запомните, что множественное наследование считается небезопасным. Обратите внимание, во многих языках программирования нет возможности реализовать такой тип наследования. Отметьте для себя опасные моменты множественного наследования и способы их избежать. Не забывайте про практическую реализацию приведенных в учебных пособиях примеров.

Рекомендуемые источники

Ашарина, И. В. Объектно-ориентированное программирование в C++: лекции и упражнения : учебное пособие / И. В. Ашарина. – 2-е изд., перераб. и доп. – Москва : Горячая линия-Телеком, 2017. – 336 с. – ISBN 978-5-9912-0423-1. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/119830>. – Режим доступа: для авториз. пользователей. Стр. 249 – 253.

Объектно-ориентированное программирование на C++ : учебник / И. В. Баранова, С. Н. Баранов, И. В. Баженова [и др.]. – Красноярск : СФУ, 2019. – 288 с. – ISBN 978-5-7638-4034-6. – Текст : электронный // Лань : электронно-библиотечная система. – URL:

2.5 Подготовка к зачету с оценкой

Форма контроля по дисциплине в первом и втором семестре изучения – зачет с оценкой. Выполнение всех видов самостоятельных работ, лабораторных работ, посещение практических и лекционных занятий – гарантия успешной сдачи зачета.

Для подготовки к зачету рекомендуется повторить темы, вынесенные на зачет. При подготовке обращайтесь не только к конспектам лекций, но и к рекомендованным преподавателем источникам. Организуйте план повторения материала таким образом, чтобы каждый день прорабатывать примерно одинаковый по объему материал. Изучая учебники и учебные пособия, отвечайте на контрольные вопросы. Прорешайте задачи примерного билета. Если после изучения материала Вы не смогли найти ответы на какие-либо вопросы – посетите консультацию перед зачетом, кроме ответов на вопросы по теме зачета на консультации освещаются организационные вопросы проведения зачета, время начала зачета, время проведения зачета, план проведения зачета и т.д.

Пример билета (1 семестр)

Билет №1

1. Цикл `for` в языке Си. Синтаксис. Принцип работы. Запишите с помощью цикла `for` фрагмент программы, выводящий на экран значения 2,4,8,10,12.

2. Запишите алгоритм, вычисляющий сумму цифр заданного числа, используя блок-диаграмму.

3. Напишите программу, которая запрашивает с клавиатуры размерность массива, задает элементы массива случайным образом и выводит их на экран. Найдите и выведите на экран индекс максимального элемента.

Пример билета (2 семестр)

Билет №1

1. Принципы объектно-ориентированного программирования. Свойство инкапсуляции. Приведите пример инкапсуляции.

2. Понятие класса в Си++. Конструкторы.

3. Допisać в класс `Stack`

а) метод, возвращающий первый элемент стека (его информационное значение) без удаления,

б) перегрузку оператора `=`.

3 РЕКОМЕНДУЕМЫЕ ИСТОЧНИКИ

1. Пермякова, Н.В. Информатика и программирование: Учебное пособие [Электронный ресурс] / Н.В. Пермякова – Томск: ТУСУР, 2016. – 188 с. – Режим доступа: <https://edu.tusur.ru/publications/7678>,
2. Подбельский, В. В. Курс программирования на языке Си : учебник / В. В. Подбельский, С. С. Фомин. – Москва : ДМК Пресс, 2012. – 384 с. – ISBN 978-5-94074-449-8. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/4148>. – Режим доступа: для авториз. пользователей.
3. Потопахин, В. Искусство алгоритмизации [Электронный ресурс] / В. Потопахин. – Электрон. дан. – Москва : ДМК Пресс, 2011. – 320 с. – Режим доступа: <https://e.lanbook.com/book/1269> . – Загл. с экрана.
4. Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона [Электронный ресурс] : учебное пособие / Н. Вирт. – Электрон. дан. – Москва : ДМК Пресс, 2010. – 272 с. – Режим доступа: <https://e.lanbook.com/book/1261>. – Загл. с экрана.
5. Златопольский, Д.М. Подготовка к ЕГЭ по информатике. Решение задач по программированию [Электронный ресурс] : учебное пособие / Д.М. Златопольский. – Электрон. дан. – Москва : ДМК Пресс, 2017. – 252 с. – Режим доступа: <https://e.lanbook.com/book/100911>. – Загл. с экрана.
6. Ашарина, И. В. Объектно-ориентированное программирование в C++: лекции и упражнения : учебное пособие / И. В. Ашарина. – 2-е изд., перераб. и доп. – Москва : Горячая линия-Телеком, 2017. – 336 с. – ISBN 978-5-9912-0423-1. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/119830>. – Режим доступа: для авториз. пользователей.
7. Аммерааль, Л. STL для программистов на C++ / Л. Аммерааль. – Москва : ДМК Пресс, 2006. – 240 с. – ISBN 5-89818-027-3. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/1218>. – Режим доступа: для авториз. пользователей.
8. Объектно-ориентированное программирование на C++ : учебник / И. В. Баранова, С. Н. Баранов, И. В. Баженова [и др.]. – Красноярск : СФУ, 2019. – 288 с. – ISBN 978-5-7638-4034-6. – Текст : электронный // Лань : электронно-библиотечная система. – URL: <https://e.lanbook.com/book/157572>. – Режим доступа: для авториз. пользователей.

ПРИЛОЖЕНИЕ А

Темы и примерные варианты контрольных работ

1. Синтаксис и алфавит языка Си

Вариант 1		
<p>1. Выберите тип передачи управления, использующийся в структурном программировании:</p> <ul style="list-style-type: none"> • безусловная передача • условная передача • функционально-зависимая передача 	<p>Посчитайте количество лексем в представленном фрагменте программы:</p> <pre>float x,y,z; printf(" -->");</pre>	<p>Выберите ключевые слова Си:</p> <ul style="list-style-type: none"> • if • while • main • factorial • integer

2. Основные типы данных. Условный оператор

Вариант 1		
<p>Опишите переменную x как указатель на тип float.</p>	<p>Что будет храниться по адресу y, если выполниться фрагмент программы:</p> <pre>int *y; int k = 12; y = &k;</pre>	<p>Что будет выведено на экран при выполнении следующего фрагмента программы:</p> <pre>int x = 7; int y = 9; int z = 0; if (x>y) { z = y*2; y = x*4; } else { z = x*2; x = y+x;} printf (" %d %d %d", x,y,z);</pre>

3. Циклы в языке Си

Вариант 1 Фамилия _____ гр _____		
<p>1. Используя цикл while, запишите фрагмент программы, который выводит на экран числа 2 5 8 11 14 17 20. Описание использованных переменных обязательно.</p>	<p>2. Что будет выведено на экран при выполнении следующего фрагмента программы:</p> <pre>int i = 25; do{ printf("%3d",i); i-=2; } while(i>=13);</pre>	<p>3. Запишите фрагмент программы, решающей следующую задачу (используйте цикл for):</p> <p>Вывести на экран числа от 0 до 12 с шагом 0.25. Фрагмент обязательно должен содержать описания использованных переменных.</p>

4. Программирование циклических процессов

Вариант 1

1. Напечатать таблицу кубов целых чисел от a до $a+10$ в виде:

```
*****
* a *   куб *
*****
*   *   *
*   *   *
*   *   *
*****
```

Значение a вводить с клавиатуры.

2. Последовательность чисел a_0, a_1, a_2, \dots образуется по закону: $a_0=1, a_k = ka_{k-1} + 1/k (k = 1, 2, \dots)$. Дано натуральное число n . Получить a_1, a_2, \dots, a_n .

5. Массивы в языке Си

Вариант 1 Фамилия _____ гр _____		
<p>1. Что будет выведено на экран при выполнении следующей программы:</p> <pre>int main(int argc, char *argv[]) { system("chcp 1251"); int x[10] = {2,7,6,1,9,5,8,3,4,0}; int k = 0,i; for (i=0;i<10;i++) if (x[i]%2==0) printf("%3d",i); printf("\n"); system("PAUSE"); return EXIT_SUCCESS; }</pre>	<p>2. Что будет выведено на экран при выполнении следующей программы:</p> <pre>int main(int argc, char *argv[]) { system("chcp 1251"); int x[10] = {2,7,6,1,9,5,8,3,4,0}; int k = x[0],i; for (i=1;i<10;i++) if (x[i]>k) k = x[i]; printf("%3d",k); printf("\n"); system("PAUSE"); return EXIT_SUCCESS; }</pre>	<p>3. Что будет выведено на экран при выполнении следующей программы:</p> <pre>int main(int argc, char *argv[]) { system("chcp 1251"); int x[10] = {2,7,6,1,9,5,8,3,4,0}; int i,j,k; int m = 3; for (j=0;j<m;j++){ k = x[9]; for (i=9;i>0;i--){ x[i] = x[i-1]; x[0] = k; } } for (i=1;i<10;i++) printf("%3d",x[i]); printf("\n"); system("PAUSE"); return EXIT_SUCCESS; }</pre>

6. Классы и объекты

6.1 Выберите названия для элементов класса:

```
class A {
private: [атрибут доступа, поле, метод конструктор, деструктор]
int a; int x; [атрибут доступа, поле, метод конструктор, деструктор]
public: [атрибут доступа, поле, метод конструктор, деструктор]
A(int a1, int x1); [атрибут доступа, поле, метод конструктор, деструктор]
A(); [атрибут доступа, поле, метод конструктор, деструктор]
float func1(); [атрибут доступа, поле, метод конструктор, деструктор]
int func2(int p); [атрибут доступа, поле, метод конструктор, деструктор]
~A();[атрибут доступа, поле, метод конструктор, деструктор]
}
```

6.2 Выберите принципы объектно-ориентированного программирования:

- а) наследование
- б) инкапсуляция
- в) декомпозиция
- г) видимость
- д) прозрачность

6.3 Класс A описан следующим образом:

```
class A{
protected:
int a;
public:
A(int k){ a = k-1;}
A(int k,int p){ if (k>p) a = k; else a = p;}
}
```

Выберите объект, поле a которого будет равно 0.

- а) A obj(-1,0);
- б) A obj(1);
- в) A obj(7,0);
- г) A obj(0);
- д) A obj(0,7);

7. Элементы класса

1. Какие из перечисленных ниже утверждений являются верными?

- а) Конструктор вызывается при создании объекта класса.
- б) Деструктор вызывается при удалении объекта.
- в) Конструктор не имеет возвращаемого значения.
- г) Если в классе нет конструктора, то создать объект такого класса нельзя.
- д) Деструктор класса возвращает ссылку на объект.
- е) Конструктор вызывается при удалении объекта.

2. Закончите предложенные ниже высказывания.

Функция, реализация которой выполнена в описании класса называется ...

Функция, описанная как friend называется ...

8. Массивы объектов класса

8.1 При запуске следующей программы:

```
1 #include <iostream>
2 using namespace std;
3 class A{
4     private:
5         int key;
6     public:
7         A() {key=0; cout << "Конструктор без параметров"<< endl;};
8         A(int i){key = i; cout << "Конструктор с параметрами"<< endl;};
9         int getKey(){ return key;};
10 };
11
12 int main() {
13     system("chcp 1251");
14     A x[10];
15 }
```

- а) Будет создан статический массив статических объектов
- б) На экран 10 раз выведется строка "Конструктор без параметров"
- в) Будет создан статический массив динамических объектов
- г) Будет создан динамический массив статических объектов
- д) Будет создан динамический массив динамических объектов
- е) На экран 10 раз выведется строка "Конструктор с параметрами"

8.2 Вставьте в программу описание статического массива динамических объектов класса B:

```
#include <iostream>
#include <cstdlib>
using namespace std;
class B{
private:
int b;
public:
```

```

        B() {b=0;};
        B(int i){b = i;};
        int getb(){ return b;};
    };

    int main() {
        system("chcp 1251");
        ...
    }

```

9. Класс array

Что будет выведено на экран при выполнении следующей программы?

```

9  #include <iostream>
10 #include <array>
11
12 using namespace std;
13
14 int main()
15 {
16
17     array <int,5> x = {2,9,3,1,4};
18     array <int,5>::iterator j = x.begin();
19     array <int,5>::value_type k = *j;
20     for(auto &i:x){
21         if(i<k) k = i;
22     }
23     cout << k;
24     return 0;
25 }

```

10. Наследование

10.1 Дан код программы:

```

kr4.cpp
1 #include <iostream>
2 using namespace std;
3 /* run this program using the console pauser or add your own getch, sy
4 class A{
5     protected:
6         int key;
7     public:
8         A(int i=0){cout << "Конструктор класса A"<< endl;
9             key = i;};
10        ~A() { cout << "Объект класса A удален"<<endl;};
11        void show(){
12            cout << "Поле key объекта класса A " << key << endl;};
13    };
14 class B : public A{
15     protected:
16         int keyB;
17     public:
18         B(int i=0):A(i+1){
19             cout << "Конструктор класса B"<< endl;
20             keyB = i; };
21         ~B() { cout << "Объект класса B удален"<<endl;};
22         void show(){
23             cout << "Поле key объекта класса A " << key << endl;
24             cout << "Поле keyB объекта класса B " << keyB << endl; };
25    };
26 int main(int argc, char** argv) {
27     system("chcp 1251");
28     A object(4);
29     B object1(5);
30     A* object2 = new A(6);
31     A* object3 = new B(7);
32     object.show();
33     object1.show();
34     object2->show();
35     object3->show();
36     delete object2;
37     delete object3;
38     return 0;
39 }

```

Вставьте пропущенные слова таким образом, чтобы утверждения были истинными.

Класс А - ... для класса В.

Класс В - ... от класса А .

Для объекта object1 будет выполняться метод show из класса ...

В программе реализован ...

При создании object1 ...

10.2

```
4 using namespace std;
5 class A{
6     private: int a;
7     protected: int b;
8     public: int c;
9     A(int x=0, int y=0, int z=0){
10        a=x; b=y; c=z;
11    }
12    void show(){
13        cout << a << " " << b << " " << c << " ";
14    }
15 };
16 class B : public A{
17     public:
18     B(int x=0, int y=0, int z=0) : A(x+1,y+1,z+1){}
19 };
20
21 int main(int argc, char** argv) {
22     A obj1;
23     B obj2(1,2,3);
24     obj1.show();
25     obj2.show();
26 }
```

Ответьте на следующие вопросы:

Какой тип наследования использован в приведенном примере?

2. Что будет выведено на экран?

3. В классе В поле а ...

4. В классе В поле b ...

5. В классе В поле с ...