Методические материалы к кейсу по курсу «Микроэлектроника и схемотехника»

Что проверяем
(перечень конкурсных кодификаторов / тем элективных курсов)

При проверке важно уделить внимание следующим пунктам

- Использование всех перечисленных в задании элементов. Все перечисленные в задании элементы должны быть подключены и задействованы в схеме.
- Правильность сборки схемы устройства, аккуратность соединения элементов. Проверяется правильность подключения элементов и читаемость схемы.
- Приведение алгоритма работы программы в виде текста или блок-схемы. Алгоритм не должен содержать логических ошибок. Возможно приведение алгоритма в виде подробного описания работы устройства либо в виде блок-схемы.
- Использование в коде программы директив define, условий, циклов, функций, комментариев для пояснения работы программы. В написанном коде может быть использована часть вышеперечисленного, однако, для получения максимального балла за этот критерий необходимо написать собственную функцию, реализующую часть программы.
- Приведение полного кода программы. Отсутствие ошибок в коде. Код структурирован, оформлен в едином стиле. В ответе на билет должен быть приведен полный код программы с учетом вышеперечисленных требований и рекомендаций. Под структурированием и оформлением в едином стиле кода программы подразумевается наличие отступов (табуляции).

Что учащийся должен знать/уметь

Знать

1. Основы проектирования электрических схем.
2. Структуру кода на языке программирования Arduino C.
3. Базовые функции языка программирования Arduino C.
4. Принципы работы с аналоговыми и цифровыми элементами (ультразвуковой дальномер, пьезоэлемент, сервопривод, датчик температуры, фоторезистор, потенциометр, светодиод, резистор, кнопка).
5. Правила построения блок-схем.
6. Правила оформления программного кода.

Уметь

1. Определять номинал резистора по цветовой маркировке.
2. Писать программный код, реализующий задуманный алгоритм.
3. Проектировать электрические схемы в программной среде Autodesk Tinkercad.

Текстовый разбор задания демонстрационного варианта

Для решения задания рекомендуется использовать программную среду Autodesk Tinkercad. Для возможности работы необходимо зарегистрироваться на сервисе с помощью одного из трех предлагаемых вариантов. После регистрации откроется главная страница, показанная на рисунке 1.

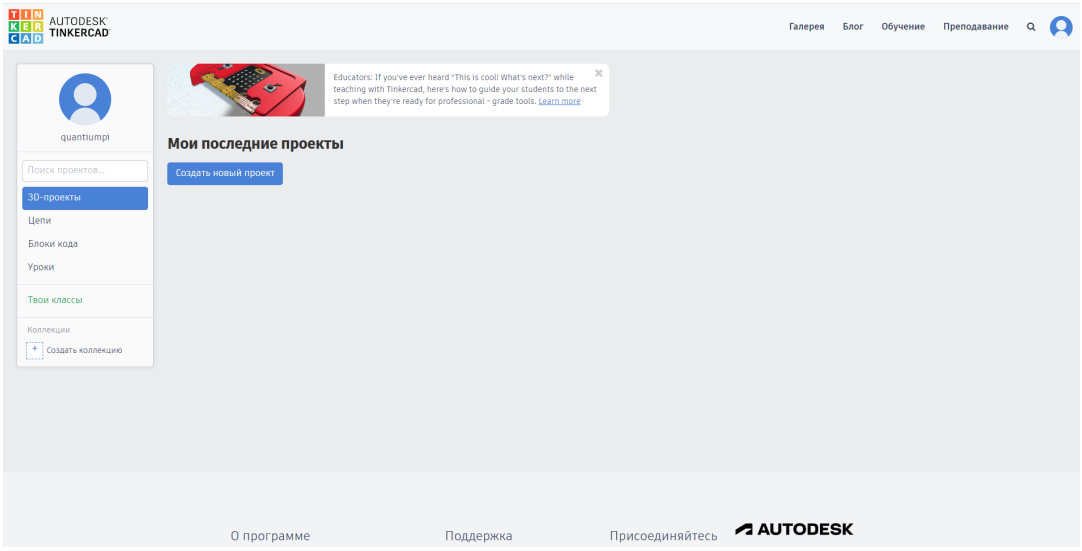

Рис. 1 – Главная страница Autodesk Tinkercad.

На главной странице открывается раздел «3D-проекты». Для создания электрических цепей необходимо перейти в раздел «Цепи». Для начала проектирования требуется нажать на кнопку «Создать цепь», как показано красной стрелкой на рисунке 2.

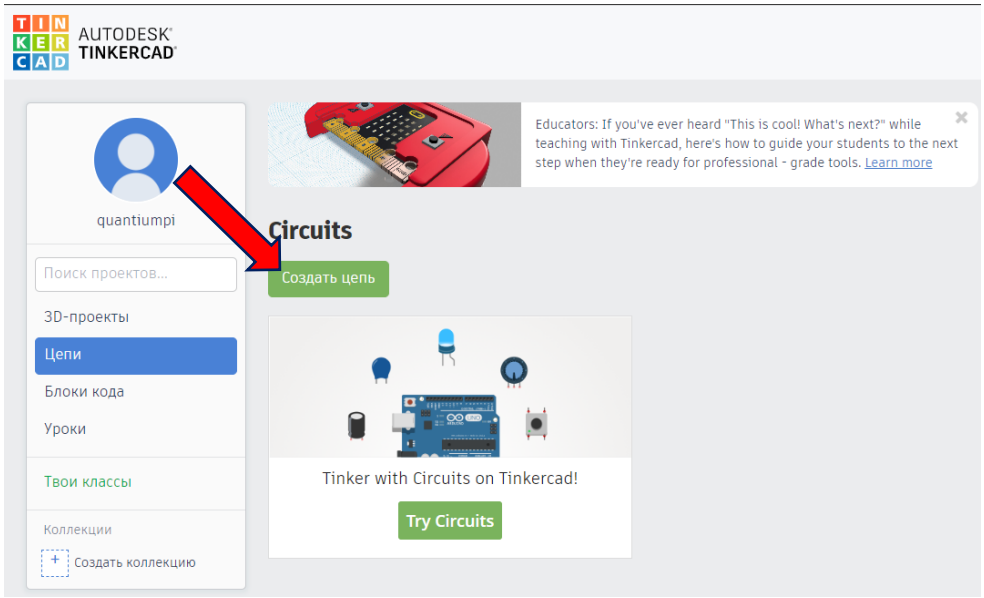

Рис. 2 – Раздел «Цепи».

После этого откроется область создания цепей, показанная на рисунке 3. В правой части экрана находится панель с компонентами. Для использования компонента его необходимо перетащить на рабочую область.

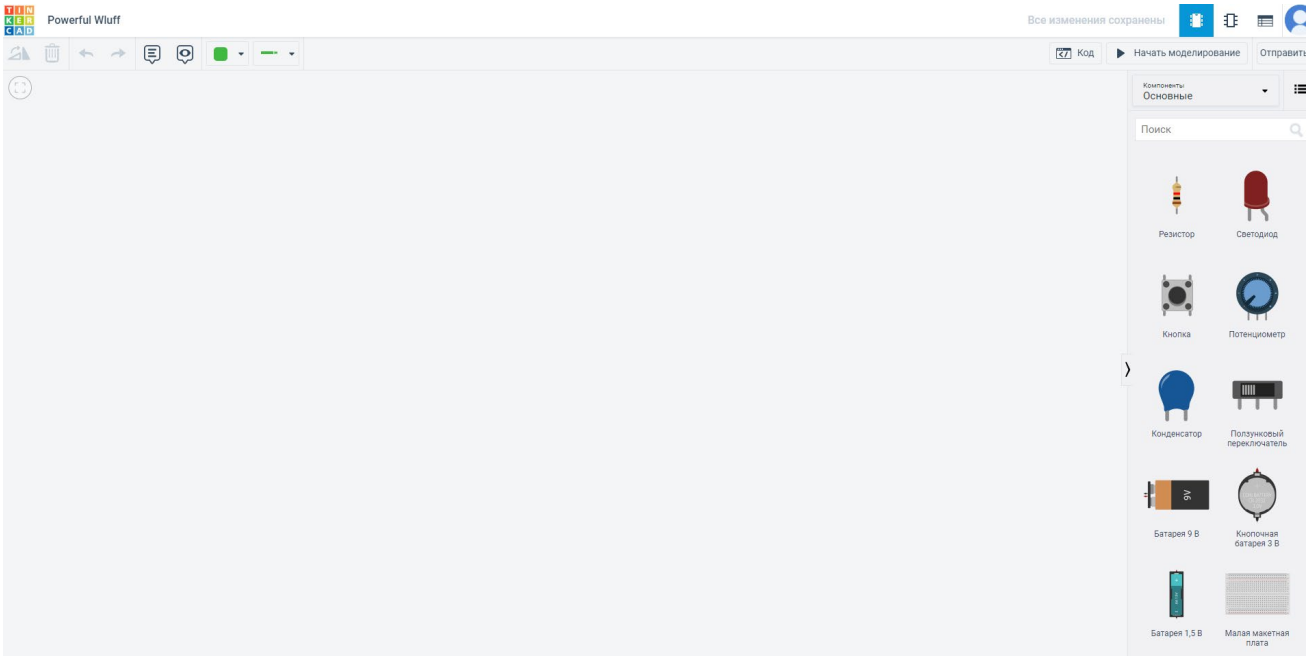

Рис. 3 – Рабочая область для создания цепей.

Над рабочей областью слева располагается панель инструментов, с помощью которой можно совершать следующие действия:

Действие	**Кнопка на панели инструментов**	**Горячие клавиши**
Вращать компонент		Клавиша «R»
Удалять		Клавиша «Delete»
Отменить действие		Клавиши «CTRL+Z»
Повторить действие		Клавиши «CTRL+Y»
Создать примечание		–
Переключение видимости примечаний		Клавиши «SHIFT+N»
Изменение цвета провода		Клавиши «1/2/3/4/5/6/7/8/9/0»

Изменение типа провода		Клавиши «SHIFT+1/2/3/4»
Дополнительные горячие клавиши		
Копировать		Клавиши «CTRL+C»
Вставить скопированный элемент		Клавиши «CTRL+V»

Для примера перенесем на рабочую область элементы демонстрационного варианта (рис. 4): Arduino UNO, макетную плату, 5 светодиодов, ультразвуковой дальномер HC-SR04, 5 резисторов.

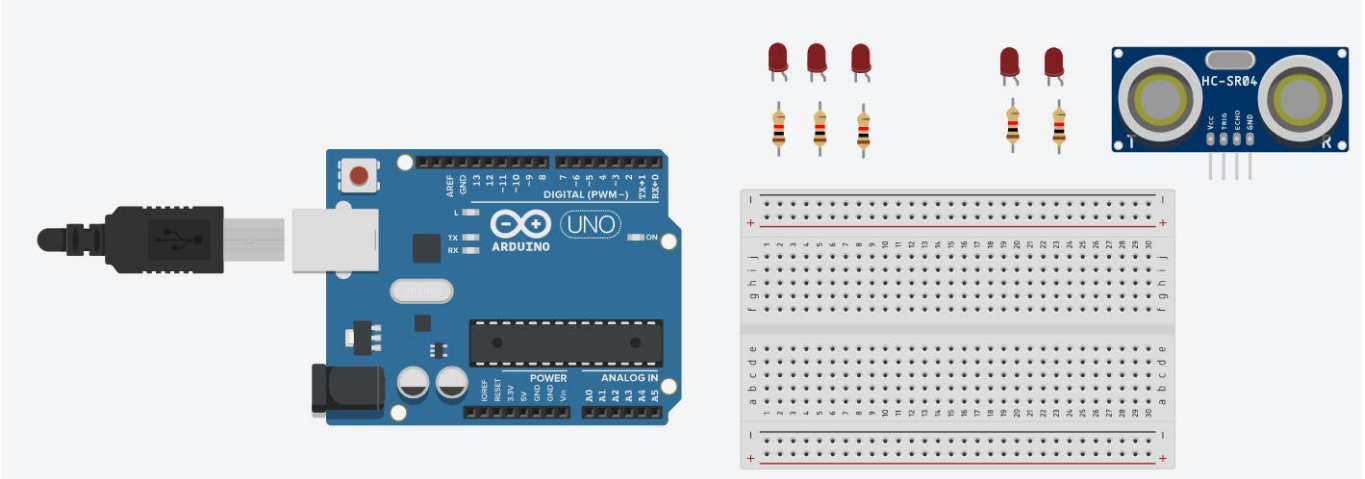

Рис. 4 – Рабочая область с расположенными элементами.

Для изменения цвета светодиода необходимо нажать на него левой кнопкой мыши. Сверху откроется дополнительное окно, как показано на рисунке 5.

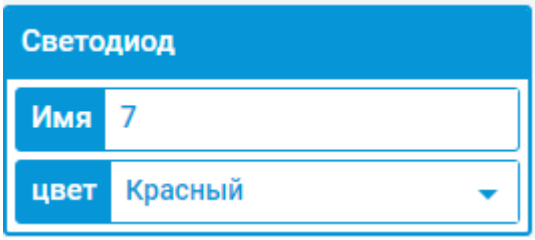

Рис. 5 – Окно параметров светодиода.

В строке «Имя» можно поменять название светодиода, в строке «цвет» выбирается цвет светодиода.

Для изменения номинала резистора необходимо нажать на него левой кнопкой мыши. Сверху появится дополнительное окно, показанное на рисунке 6.

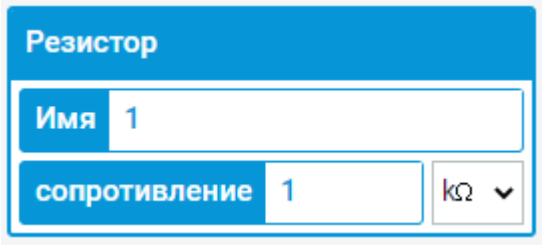

Рис. 6 – Окно параметров резистора.

В строке «Имя» можно поменять название резистора, в строке «сопротивление» выбирается необходимый номинал и из списка выбирается множитель номинала резистора.

На рисунке 7 показана рабочая область с измененными согласно заданию параметрами светодиодов и резисторов.

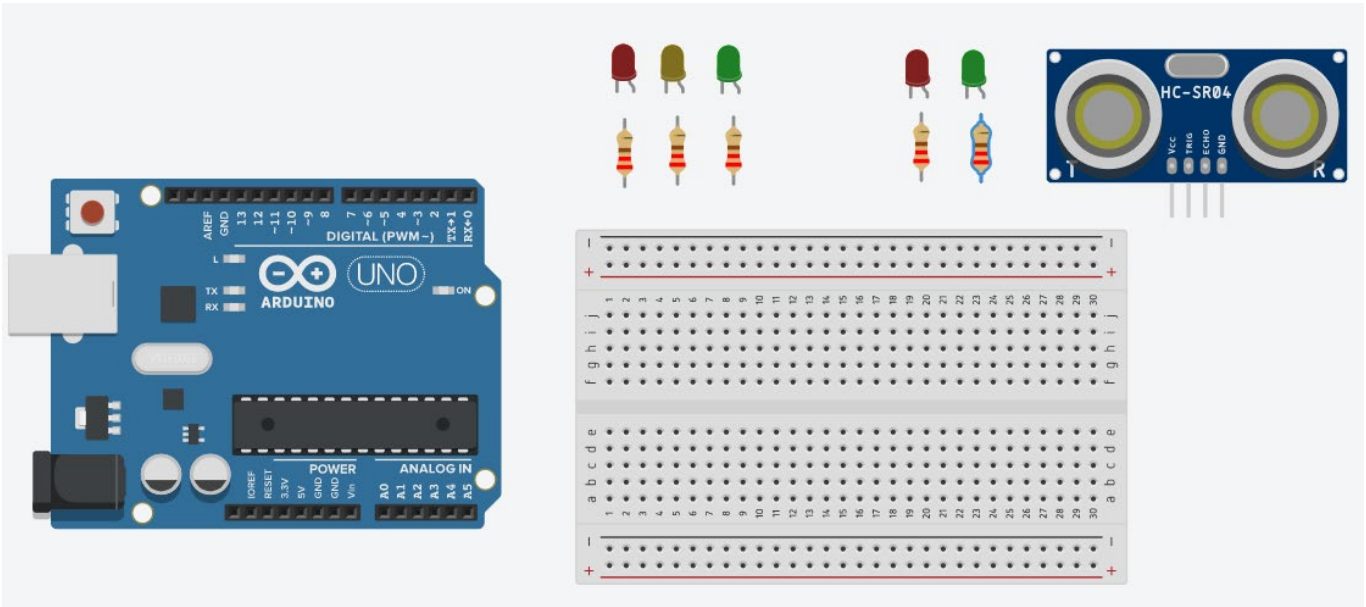

Рис. 7 – Рабочая область с измененными параметрами компонентов.

Скрытое соединение контактов макетной платы можно увидеть путем наведения курсора на монтажное отверстие. Связанные между собой контактные точки подсвечиваются, как показано на рисунке 8.

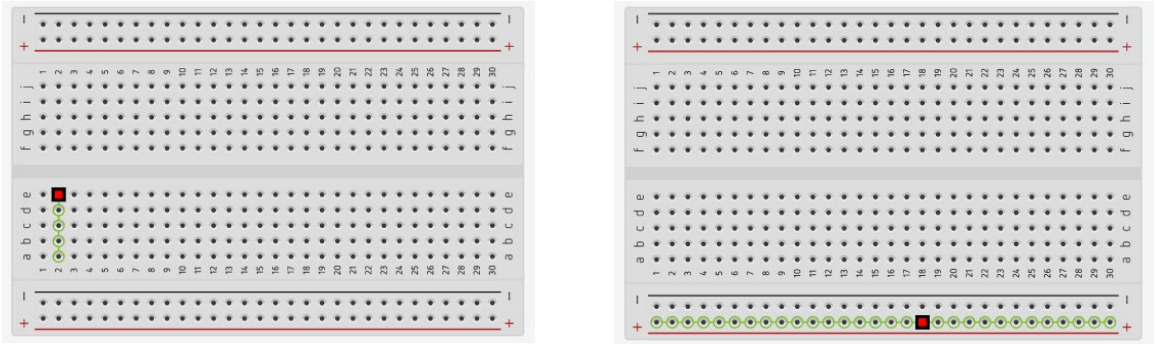

Рис. 8 – Связанные контакты макетной платы.

Расположим на макетной плате необходимые компоненты для сборки схемы, как показано на рисунке 9.

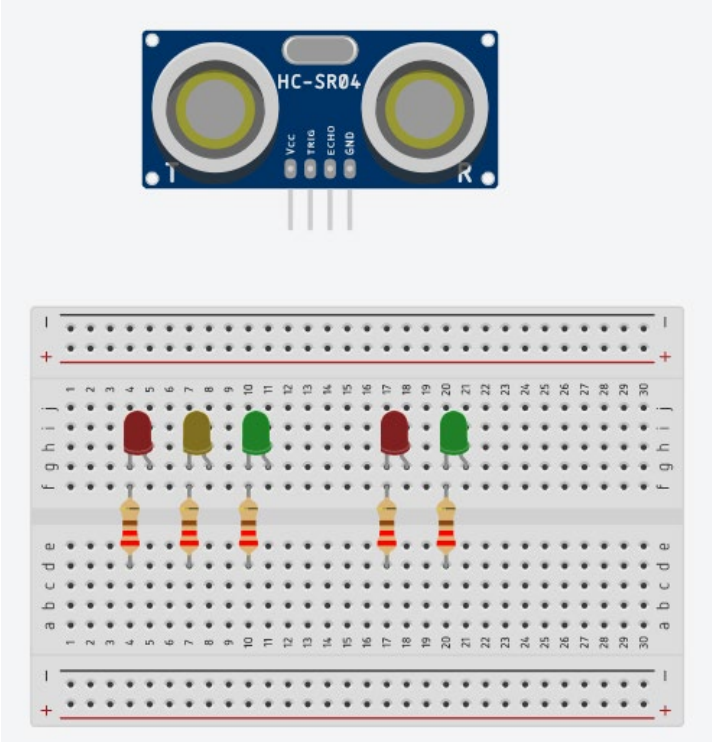

Рис. 9 – Расположение элементов на макетной плате.

На следующем этапе с помощью проводов необходимо соединить элементы между собой и с платой Arduino UNO. Для построения провода нужно левой кнопкой мыши нажать на контакт элемента, Arduino UNO или контактную точку на макетной плате. Затем необходимо подвести курсор ко второй точке соединения и нажать левую кнопку мыши. Для более аккуратного соединения можно добавить точку изгиба с помощью двойного нажатия на провод. Рекомендуется использовать привязки к

горизонтали и вертикали. Пример аккуратного соединения элементов представлен на рисунке 10.

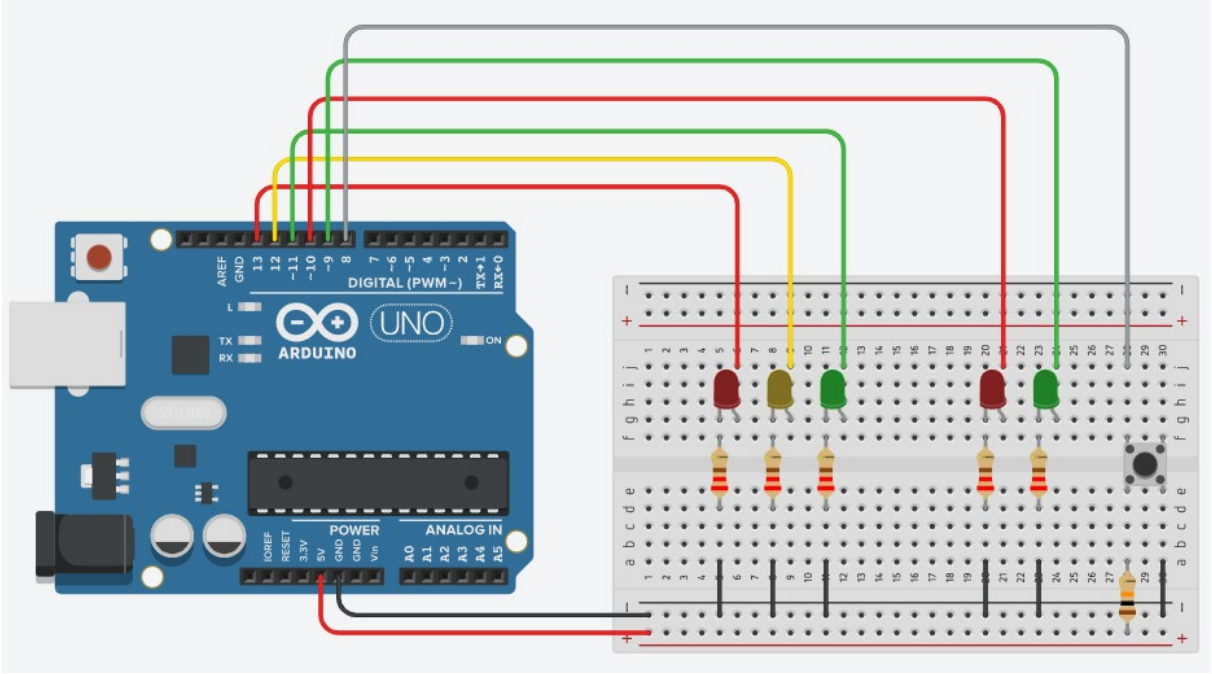

Рис. 10 – Пример аккуратного соединения элементов.

Рассмотрим контакты платы Arduino UNO, приведенной на рисунке 11. Основные контакты (пины), которые потребуются для решения задания: цифровые пины, расположенные в рамке 1; аналоговые пины, расположенные в рамке 2; пины питания, расположенные в рамке 3.

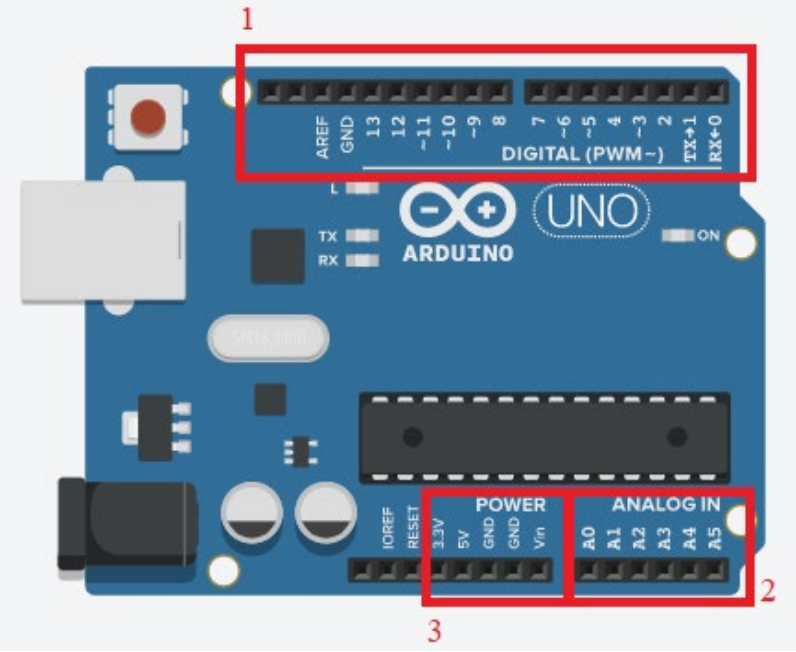

Рис. 11 – Плата Arduino UNO.

Рассмотрим подробнее схему подключения ультразвукового дальномера HC-SR04, приведенного на рисунке 12. Работа модуля основана на принципе эхолокации. Модуль посылает ультразвуковой сигнал и принимает его отражение от объекта. Измерив время между отправкой и получением импульса, можно вычислить расстояние до препятствия. Модуль подключается четырьмя проводами. Контакты VCC (+5В) и GND служат для подключения питания, а Trig и Echo— для отправки и приема сигнала дальномера. Подключим их к пинам 3 и 2 соответственно.

Рис. 12 – Ультразвуковой дальномер HC-SR04.

Для того чтобы инициализировать отправку сигнала дальномером, необходимо подать высокий сигнал длительностью 10 μs на пин Trig. После получения высокого сигнала длительностью 10 μs на пин Trig, модуль генерирует пучок из восьми сигналов частотой 40 кГц и устанавливает высокий уровень на пине Echo. После получения отраженного сигнала модуль устанавливает на пине Echo низкий уровень. Зная продолжительность высокого сигнала на пине Echo можно вычислить расстояние, умножив время, которое потратил звуковой импульс, прежде чем вернулся к модулю, на скорость распространения звука в воздухе (340 м/с). Функция pulseIn позволяет узнать длительность импульса в μs. Запишем результат работы этой функции в переменную duration.

Можно вычислить расстояние переведя скорость из м/с в см/мкс:

$$distance = duration * 340\ м/с = duration * 0.034\ м/мкс. \quad (1)$$

Далее требуется преобразовать десятичную дробь в обыкновенную:

$$distance = duration * 1/29 = duration / 29. \quad (2)$$

Поскольку звук преодолевает расстояние до объекта и обратно, необходимо поделить полученный результат на 2:

$$distance = duration / 58. \quad (3)$$

После сборки схемы, показанной на рисунке 13 можно перейти к написанию кода.

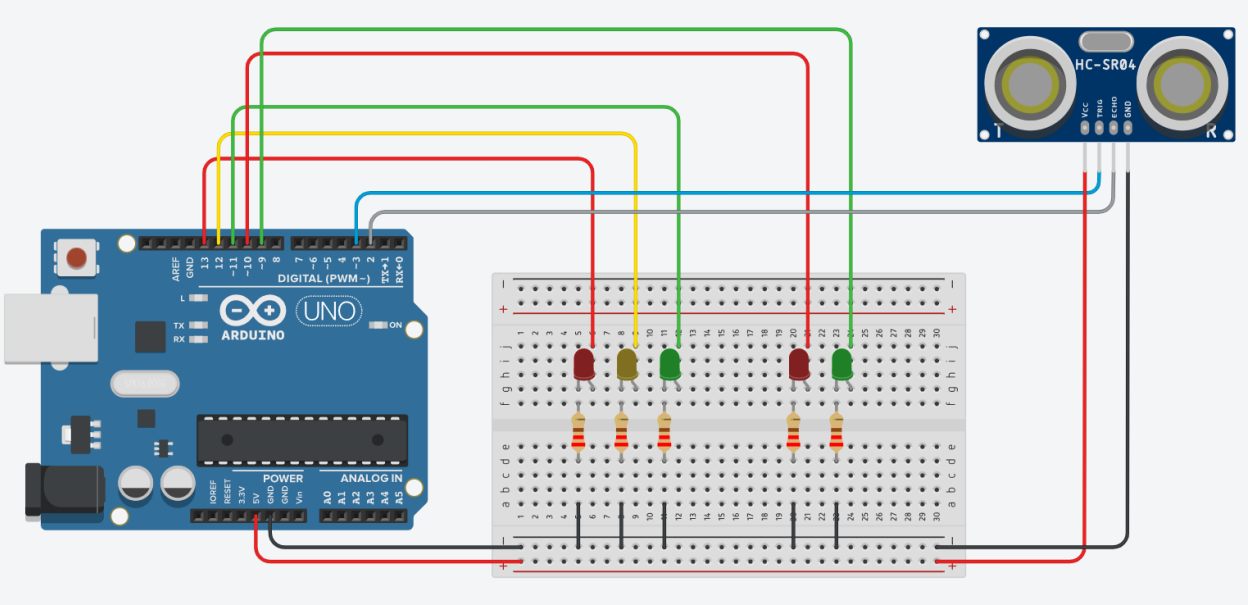

Рис. 13 – Собранная схема устройства.

Согласно критериям оценивания, требуется использование директивы define. Введем директивы, указывающие номер пина подключенных светодиодов и пинов ультразвукового дальномера:

```
# define auto_green 11
# define auto_yellow 12
# define auto_red 13
# define people_green 9
# define people_red 10
# define trigPin 3
# define echoPin 2
```

Согласно критериям оценивания, также требуется создать дополнительную функцию. Напишем функцию, отвечающую за трехкратное мигание зелеными светодиодами. В функции передается переменная типа int, отвечающая за пин светодиода.

```
void blink(int blink_LED)
{
```

Задаем цикл с переменной i типа int, изменяющейся от 0 до 2 с шагом один для реализации счетчика.

```
 for (int i=0; i<3; i++)
   {
```

Пишем алгоритм для однократного включения и выключения светодиода, затем код, написанный в фигурных скобках будет выполняться трижды циклом for.

```
    digitalWrite(blink_LED, LOW); //выключение светодиода
    delay(1000);// задержка на 1 секунду
    digitalWrite(blink_LED, HIGH); //включение светодиода
    delay(1000);// задержка на 1 секунду
   }
}
```

В функции setup необходимо инициализировать пины, к которым подключены светодиоды и ультразвуковой дальномер.

```
void setup()
{
 // Определение режимов работы выводов пинов для светодиодов
 pinMode(auto_red, OUTPUT);
 pinMode(auto_yellow, OUTPUT);
 pinMode(auto_green, OUTPUT);
 pinMode(people_red, OUTPUT);
 pinMode(people_green, OUTPUT);

// Определение режимов работы пинов для ультразвукового дальномера
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
```

Включение монитора последовательного интерфейса для возможности отладки в случае необходимости.

```
 Serial.begin (9600);
}
```

Пишем алгоритм программы для цикличной функции loop.

```
void loop()
{
```

Задаем начальные условия работы светофора. Включаем зеленый светодиод для автомобилистов и красный для пешеходов.

```
  digitalWrite(people_red, HIGH); // Начальные условия работы светофора
  digitalWrite(auto_green, HIGH); // Начальные условия работы светофора
```

Для работы с ультразвуковым дальномером необходимо сгенерировать и принять отраженную ультразвуковую волну. Конвертирование данных с дальномера в расстояние до объекта (пешехода) описано формулами (1), (2), (3).

```
   int duration, cm; //объявление переменной типа int
  digitalWrite(trigPin, LOW); //выключение пина trigPin
  delayMicroseconds(2); //приостановка выполнения программы на 2мкс
  digitalWrite(trigPin, HIGH); //включение пина trigPin
  delayMicroseconds(10); // приостановка выполнения программы на 10мкс
  digitalWrite(trigPin, LOW); //выключение пина trigPin
/*
Присвоение переменной duration значения, выдаваемого функцией pulseIn;
Функция pulseIn используется для считывания длительности импульса
сигнала заданного уровня на заданном пине
*/
  duration = pulseIn(echoPin, HIGH);
  cm = duration / 58; //перевод значения duration в сантиметры
```

Сравниваем расстояние до объекта (пешехода) с заданным пороговым значением, равным одному метру. Если расстояние больше порогового значения, то светодиоды продолжают гореть в том же режиме, что и начальные условия.

```
 if (cm > 100) // сравнение значения расстояния с пороговым значением
 {
  digitalWrite(people_red, HIGH); // действие, в случае выполнения условия
  digitalWrite(auto_green, HIGH); // действие, в случае выполнения условия
 }
```

Если расстояние меньше или равно одному метру, то выполняется условие else и идет смена сигнала светофора.

```
 else // действия, в случае невыполнения условия
 {
  //Смена светофора на разрешающий для пешеходов
  blink(auto_green); //вызов функции для мигания зеленого светофора для автомобилей
  digitalWrite(auto_green, LOW); //выключение светодиода
  digitalWrite(auto_yellow,HIGH); //включение светодиода
  delay(1000); //задержка на 1 секунду
```

```
  digitalWrite(auto_yellow,LOW); //выключение светодиода
  digitalWrite(auto_red,HIGH); //включение светодиода
  delay(1000); //задержка на 1 секунду
  digitalWrite(people_red,LOW); //выключение светодиода
  digitalWrite(people_green,HIGH); //включение светодиода
  delay(15000); //задержка на 15 секунд

  // Смена светофора на разрешающий для автомобилей
  blink(people_green); //вызов функции для мигания зеленого светофора для пешеходов
  digitalWrite(people_green,LOW); //выключение светодиода
  digitalWrite(people_red, HIGH); //включение светодиода
  delay(1000); //задержка на 1 секунду
  digitalWrite(auto_red,LOW); //выключение светодиода
  digitalWrite(auto_green, HIGH); //включение светодиода
  delay(15000); //задержка на 15 секунд для работы зеленого светофора автомобилей
 }
}
```

По критериям оценивания необходимо привести алгоритм работы программы в виде текста или блок-схемы. При описании алгоритма в виде текста должны быть выполнены следующие рекомендации: описана схема подключения, пояснены введенные директивы, описаны дополнительные функции, основные функции программы, циклы и условия. Пример описания приведен ниже:

Светодиоды подключены через резисторы 220 Ом к цифровым пинам 13, 12, 11, 10, 9. Контакты Echo и Trig ультразвукового дальномера подключены к цифровым пинам 2 и 3 соответственно. Для удобства введены директивы auto_green, auto_yellow, auto_red, people_green, people_red, button, которые задают номера соответствующих пинов. Приставка auto обозначает, что светодиоды предназначены для автомобилей; приставка people – для пешеходов; green, yellow, red – цвета светодиодов (светофора); trigPin, echoPin – пины ультразвукового дальномера.

Введена функция blink, содержащая цикл, отвечающий за мигание светодиода. В нее передается переменная типа int, соответствующая номеру пина с подключенным светодиодом.

В функции setup определяются режимы работы пинов. Для светодиодов и Trig пина дальномера – OUTPUT, т.к. они являются средством вывода информации, для пина Echo – INPUT, поскольку этот пин является средством ввода информации.

В функции loop описывается алгоритм работы светофора. Изначально задаем зеленый свет для автомобилей и красный для пешеходов. Для работы устройства сравнения генерируется и принимается ультразвуковая волна дальномера, после чего рассчитывается расстояние до

пешехода. Если расстояние до пешехода больше 1-го метра (выполнение условия if), то продолжают гореть красный светодиод для пешеходов и зеленый для автомобилей. Если пешеход приблизился на расстояние меньше 1-го метра (выполняется условие else), то сменяется сигнал светофора. Командой delay(15000) реализована задержка на 15 секунд, после чего идет переключение светофора на разрешающий свет для автомобилей, который будет гореть в течение 15-ти секунд. После чего возобновляется считывание расстояния до пешехода.

Второй вариант описания алгоритма – блок-схема. Она должна удовлетворять следующим рекомендациям: должна содержать все логические этапы выполнения программы, должна быть оформлена согласно правилам построения блок-схем. Построение возможно, как от руки, так и с помощью программного обеспечения. В случае построения блок-схемы от руки текст должен быть читаемым. Пример блок-схемы приведен на рисунке 14.

Рис. 14 – Пример блок-схемы.

Дополнительные указания для учителей по подготовке к конкурсу (на что обратить особое внимание)

Для успешного решения билетов рекомендуется обратить внимание обучающихся на следующее:

1. Схема делителя напряжения.
2. Схемы подключения аналоговых и цифровых элементов (ультразвукового дальномера, пьезоэлемента, сервопривода, датчика температуры, фоторезистора, потенциометра, светодиода, резистора, кнопки).
3. Базовые функции языка программирования Arduino C (setup, loop, digital/analogWrite, digital/analogRead, delay, map, tone и т.д.).
4. Правила построения блок-схем.
5. Правила оформления программного кода.
6. Определение номинала резисторов по цветовой маркировке.
7. Проектирование электрических схем в программной среде Autodesk Tinkercad.

Методические материалы к кейсу по курсу «Язык программирования Java»

Что проверяем (перечень конкурсных кодификаторов / тем элективных курсов)

При проверке важно уделить внимание следующим пунктам

- Использование выраженных методов. Каждая команда **должна** быть реализована в отдельном методе, обеспечивающем её функциональность.
- Корректное использование типов данных. Для числовых данных, требуемых в задании, нужно использовать соответствующие типы данных в языке программирования.
- Корректное сравнение строк. В Java строки необходимо сравнивать через метод equals() а не через оператор ==, из-за ссылочной природы строкового типа данных. В процессе изучения языка, когда на примитивных типах данных показываются примеры сравнения значений опыт использования оператора == закономерно перетекает на другие типы, и важно вовремя обратить внимание ученика на подход, базирующийся на содержимом объектов, а не на ссылочном сравнении.
- Работа с исключениями. Важно обратить внимание на работу как с проверяемыми, так и непроверяемыми исключениями. Первые требуют декларации в сигнатуре методов, и этот принцип необходимо понимать. Вторые могут привести к непредвиденным ошибкам в процессе выполнения. Например, при некорректном делении, или обращении к несуществующему объекту.
- Работа с файлами. Внешние ресурсы необходимо закрывать в обязательном порядке. Необходимо убедиться в обязательном вызове close() на объектах, предоставляющим доступ к, например, файлам.

Что учащийся должен знать/уметь

Знать

7. Принципы работы с переменными.
8. Способы хранения неопределенного заранее количества данных.
9. Способы хранения связанных данных.
10. Возможности языковой платформы (типы данных, области их значений).
11. Способы разделения алгоритма на последовательность независимых логических блоков.
12. Принципы взаимодействия языка программирования с внешними ресурсами, такими как файлы.
13. Возможности базовых структур данных (массив, список, словарь, множество).

Уметь

4. Писать программный код, реализующий задуманный алгоритм.
5. Использовать конструкции языка программирования для выразительного и целенаправленного решения алгоритмической задачи.
6. Безопасно работать с внешними ресурсами из языка программирования. Например – файлами.
7. Взаимодействовать при помощи языка программирования со стандартными потоками ввода вывода.
8. Применять базовые структуры данных для решения алгоритмических задач (массивы, списки, словари, множества).
9. Выносить программные компоненты в независимые методы.

Текстовый разбор каждого задания демо варианта

Для решения всех заданий используется единый подход, при котором берется очередная строка с входного потока данных, преобразуется в массив токенов используя пробел как разделитель, который используется в необходимом виде во всех операциях. Первый токен представляет собой команду, которую необходимо выполнить, по условию первым словом до пробела идет именно команда. Дальнейшие токены в зависимости от задания представляют соответствующие данные.

Важно отметить, что способ, применяемый в решении демо варианта не единственно верный. Например, можно использовать Scanner, и на нем при переходе на очередную строку вызывать next(), результатом будет токен, указывающий команду для выполнения. И после этого, в зависимости от токена, считывать последовательно все необходимые значения. Такой подход может быть проще в понимании для ученика, в решении демо варианта был выбран способ через метод slipt() на всей строке ввиду его компактности.

Вместо Scanner можно использовать BufferedReader в связке с InputStreamReader, если ученику такой подход удобнее. Сам способ преобразования байтов на входе в корректные типы на выходе не важен, и не влияет на итоговую оценку.

В реализации двух команд используется метод Contact parseContact(String[] tokens, int indexOffset), преобразующий токены из первого аргумента в экземпляр класса Contact. Данный метод использует 5 токенов как аргументы для конструктора класса Contact. Аргумент indexOffset используется для указания смещения индексов пяти токенов, так как при вводе с консоли они смещены вправо, первым идет токен с командой. В при чтении с файла смещения нет, так как информация о контакте записывается сразу.

INSERT <ФамилияИО> <группа> <email> <тел> <гр> – добавляет данные о новом контакте с заданными значениями.

При преобразовании считанной строки получается массив из шести токенов, в которых первым идет команда INSERT, а пять последующих представляют собой поля контакта в соответствии с заданием. Информация из массива преобразуется в

объект при помощи метода parseContact, значение аргумента indexOffset равно нулю, так как реализация использует расположение с учётом команды по умолчанию.

Полученный объект добавляется в коллекцию data. По условию все <ФамилияИО> уникальные, и ситуации, в которой будет добавлено несколько пользователей с одинаковыми <ФамилияИО> не представляется возможной. Это снимает необходимость проверять наличие такого контакта в списке, и позволяет добавить контакт в коллекцию без выполнения дополнительных действий.

ECHO <ФамилияИО> – выводит в консоль всю информацию о контакте с заданными фамилией и инициалами.

При преобразовании считанной строки получается массив из двух токенов, в которых первым идет команда ECHO, а во втором передается <ФамилияИО> контакта, информацию о котором необходимо вывести. Метод showInfo принимает на вход два аргумента data – коллекция контактов, и name – строковое значение, содержащее в себе <ФамилияИО>.

Далее при помощи цикла for находится целевой контакт, на основании вызова метода equals(). Важно использовать именно такой способ сравнивания строк, так как при использовании оператора == вероятно неверное поведение программы из-за особенностей реализации и оптимизации работы со строками в той или иной версии JDK. По условию все поля всегда имеют корректные значения, и NullPointerException не грозит, так как все строки инициализированы.

Информация о найденном контакте выводится на экран при помощи метода println(), который принимает на вход контакт как тип Object, и для преобразования к строке вызывает на нем метод toString(). Данный метод переопределен в классе Contact для форматирования информации о контакте в соответствии с заданием. Метод формирует строку, где все поля контакта разделены единственным пробелом. После вывода информации на экран выполнение цикла прерывается при помощи ключевого слова break. Данное действие не несет обязательных характер, так ка по условию <ФамилияИО> является уникальным, и двойного вывода на консоль не случится. Но принимая во внимание уникальность <ФамилияИО> нет смысла продолжать итерирование по коллекции, так как единственное <ФамилияИО> уже обработано. Кроме того, вместо использования конструкции break допустимо

использование конструкции return, которая позволит гарантированно завершить работу метода, в который по условию должна быть помещена логика обработки команды.

DEL <ФамилияИО> – удаляет контакт с заданными фамилией и инициалами.

При преобразовании считанной строки получается массив из двух токенов, в которых первым идет команда DEL, а во втором передается <ФамилияИО> контакта, информацию о котором необходимо удалить. Метод removeContact принимает на вход два аргумента data – коллекция контактов, и name – строковое значение, содержащее в себе <ФамилияИО>.

Далее при помощи цикла for находится целевой контакт, на основании вызова метода equals(). Важно использовать именно такой способ сравнивания строк, так как при использовании оператора == вероятно неверное поведение программы из-за особенностей реализации и оптимизации работы со строками в той или иной версии JDK. По условию все поля всегда имеют корректные значения, и NullPointerException не грозит, так как все строки инициализированы.

Информация о найденном контакте удаляется из коллекции при помощи метода remove(). После вывода информации на экран выполнение цикла прерывается при помощи ключевого слова break. Данное действие носит обязательных характер, так как продолжать итерирование по измененной коллекции запрещено, при работе с типом данных ArrayList. По условию <ФамилияИО> является уникальным, и единственно возможная запись уже была удалена. Потому прерывание цикла никак не скажется на логике поведения программы. Кроме того, вместо использования конструкции break допустимо использование конструкции return, которая позволит гарантированно завершить работу метода, в который по условию должна быть помещена логика обработки команды.

SORT_PRINT <номер_поля_сорт> – выводит в консоль информацию обо всех добавленных контактах, сортируя строки по заданному полю по возрастанию. Если поле строковое, то сортировка должна выполняться по алфавиту.

При преобразовании считанной строки получается массив из двух токенов, в которых первым идет команда SORT_PRINT, а во втором передается <номер_поля_сорт>, показывающее, по какому именно полю необходимо провести

сортировку. Метод sortPrint() принимает коллекцию контактов data, и номер поля для сортировки.

Первым действием является сортировка коллекции данных в порядке, обоснованном номером поля в выполняемой команде. В данной ситуации изменяется порядок следования элементов внутри коллекции, но это не является проблемой, так как ни одна другая команда по заданию не зависит от порядка следования элементов в хранилище. Сортировка выполняется при помощи метода sort() класса ArrayList. Он принимает единственным аргументов объект, реализующий интерфейс Comparator. В решении демо варианта представлен способ создания такого объекта при помощи лямбда выражения, представляющего собой экземпляр анонимного класса, реализующего единственный метод. Два входных аргумента – два произвольных с точки зрения разработчика объекта из коллекции, результатом работы должно быть целое чисто. Равное нулю, если объекты равны по некоторому критерию, отрицательное число если первый аргумент меньше второго, и положительное если первый аргумент больше, чем второй. Данная логика обеспечивается методом compare класса Contact. Данный метод принимает второй контакт для сравнения, и номер поля, по которому необходимо выполнить сравнение. Для строковых полей выполняется вызов метода compareTo() на соответствующей строке, который ведет себя в соответствии с логикой работы интерфейса Comparator, и результат работы возвращается напрямую. Для года рождения используется разница между годом рождения сравнивающего контакта и сравниваемого. Если год рождения сравнивающего контакта больше – значение получится положительное, что будет значить превосходство сравнивающего контакта, при равенстве вернётся ноль, а отрицательный результат будет означать превосходство сравниваемого контакта. Поведение метода при некорректном номере поля не регламентировано заданием, для него можно вернуть любое значение.

После выполненной сортировки все контакты последовательно выводятся на консоль при помощи метода println(), который вызывает метод toString() класса Contact.

INTO_FILE <имя_файла> – сохраняет добавленные контакты в заданный текстовый файл в формате команды INSERT (с разделением значений пробелами): <ФамилияИО> <группа> <email> <тел> <гр>.

При преобразовании считанной строки получается массив из двух токенов, в которых первым идет команда INTO_FILE, а во втором передается <имя_файла>, показывающее, по какому пути должен будет располагаться файл со всей информацией, на данный момент сохранённой в системе. Метод saveToFile() принимает коллекцию контактов data, и путь к целевому файлу.

Для записи в файл создаются два объекта. Типа FileOutputStream – предоставляет доступ к записи в файл по определенному пути. Используется как аргумент конструктора для объекта типа PrintWriter, позволяющий записывать информацию в файл при помощи инструментов, аналогичных работе с консольным выводом. Создание данных объектов помещено в конструкцию try-with-resources, обеспечивающую гарантированное закрытие файла при завершении работы с ним.

После создания вспомогательных файлов при помощи цикла for перебираются все контакты, переданные в первом аргументе метода, и последовательно записываются в файл, в соответствии с тем порядком, в котором объекты находятся в коллекции сейчас. В задании не регламентирован порядок для вывода информации в файл, потому порядок тут не играет роли, и может быть произвольным, так как зависит от вызовов команд SORT_PRINT, INSERT, DEL.

FROM_FILE <имя_файла> – загружает контактные данные из заданного текстового файла, полностью замещая текущий состав записей о контактах. Считается, что загружаемый файл был создан при вызове команды INTO_FILE.

При преобразовании считанной строки получается массив из двух токенов, в которых первым идет команда FROM_FILE, а во втором передается <имя_файла>, показывающее, по какому пути расположен файл со всей информацией, которую необходимо загрузить в систему. Метод readFromFile() принимает коллекцию контактов data, и путь к целевому файлу.

Для считывания данных из файла создаются два вспомогательных объекта. Типа FileInputStream – предоставляет поток данных, находящихся в файле. Он является аргументом для конструктора класса Scanner, который позволяет считывать данные

пользуясь подходом, ранее применяемом при работе с потоком ввода из консоли. Создание данных объектов помещено в конструкцию try-with-resources, обеспечивающую гарантированное закрытие файла при завершении работы с ним.

После создания вспомогательных классов происходит считывание данных до конца файла. Факт окончания данных обеспечивается возвратом значения false из метода hasNext() класса Scanner. При считывании очередной строки из файла она разбивается на токены, используя пробельный символ как разделитель. Токены передаются в метод parseContact() для создания на их основе нового экземпляра класса Contact, indexOffset здесь равен -1 , так как данные в файле сразу начинаются с <ФамилияИО>, и необходимо сместить индексы от логики, рассчитанной на наличие команды в первом токене, влево на один индекс. Созданный экземпляр Contact добавляется в локально созданную коллекцию contacts, по завершению считывания данных из файла локально созданная коллекция контактов возвращается как результат работы метода, и замещает собой хранимую на уровне системы коллекцию data, тем самым выполняя требования команды.

Дополнительные указания для учителей по подготовке к конкурсу (на что обратить особое внимание)

- Способ разбиения логики на методы. В решении демо варианта приведен пример разделения логики на методы, но фактически для команды INSERT нет отдельного метода insertToData, или подобного, так как метод parseContact() обеспечивает практически всю работу в рамках обработки данной команды. Фактически возможно решение, в котором будет выделен отдельный метод, в котором будут две операции: преобразование токенов в сущность и добавление последней в коллекцию.
- Способ хранения информации о сущностях. В Java принято использовать классы для группировки полей в единую структуру. Важно отметить, как именно ученик хранит данные. При помощи класса, массива, словаря, списка.
- Выразительность созданных методов. Методы, отвечающие за выполнение соответствующих команд, могут называться a, b, c или выразительно отражать своё содержание. Стоит обратить внимание на способ именования языковых сущностей.

Методические материалы к кейсу по курсу «Основы программирования на Python»

Что проверяем (перечень конкурсных кодификаторов/тем элективных курсов).

Темы, знание которых проверяется на конкурсе:

- Числовые типы данных (float и int);
- Ветвление (условные инструкции);
- Циклы while и for;
- Работа с объектами логического типа (bool);
- Работа со строками;
- Работа с коллекциями (списками, кортежами, словарями);
- Функции;
- Файлы (в текстовом режиме).

При решении варианта особенно важно реализовать решение в виде набора функций. Каждая функция при решении оценивается отдельно. Пример таблицы для оценивания из демонстрационного варианта:

№ п/п	Команда	Реализована как функция и выполняется корректно	Реализована не как функция и выполняется корректно	Выполняется не корректно при любом способе реализации
1	ДОБАВИТЬ	10 баллов	5 баллов	0 баллов
2	ПОКАЗАТЬ	10 баллов	5 баллов	0 баллов
3	УДАЛИТЬ	10 баллов	5 баллов	0 баллов
4	СВОДКА	10 баллов	5 баллов	0 баллов
5	СОХРАНИТЬ	10 баллов	5 баллов	0 баллов
6	ЗАГРУЗИТЬ	10 баллов	5 баллов	0 баллов
ИТОГО		**60 баллов**	**30 баллов**	**0 баллов**

Что учащийся должен знать/уметь.

Для успешного написания конкурса учащиеся должны быть уверенными пользователями ПК, владеть навыками программирования на языке Python, знать основные синтаксические конструкции, принципы их работы и уметь их использовать. Ниже прилагается возможный план занятий для подготовки к конкурсу. В нем перечислены темы и подтемы, составляющие теоретический минимум для подготовки.

Номер занятия	Объем часов	Содержание занятия
1	2 по 45 минут	Знакомство со средой. Числовые типы данных. План: • Введение • Знакомство с IDE PyCharm • Функции ввода/вывода • Комментарии • Введение в PEP8

		• Введение в типы данных. • Операции над числами (+, -, *, /, //, **) • Приоритет операций • Функции преобразования типов (int, float) • Задачи
2	2 по 45 минут	Условная инструкция (оператор). План: • Синтаксис условной инструкции, блоки кода, отступы • and, or, сложные условия • Вложение условий, elif • Базовые операции над строками • Задачи
3	2 по 45 минут	Циклы while и for. План: • Синтаксис и использование цикла While • Синтаксис цикла for • Начальное значение, шаг и конечное значение range • Когда какой цикл использовать (while или for) • Инструкции break и continue • Задачи
4	2 по 45 минут	Логический тип данных. План: • Введение • True, False • Преобразование типов в логический тип, интерпретация типов в логическом контексте, функция bool • Флаги • Циклы с постусловием • Задачи
5	2 по 45 минут	Введение в коллекции. Строки. План: • Понятие коллекции. Строка, как коллекция • Индексация в строках • Перебор элементов строки • Хранение текстов в памяти. Введение в кодировки, Unicode • Срезы строк • Перевертывание строки (срез [::-1]) • Задачи

6	2 по 45 минут	Списки и кортежи. План: • Понятие списка, характеристики (изменямость, упорядоченность) • Создание списков • Индексация и срезы • Добавление и удаление элементов • Перебор элементов списка • Понятие кортежа, характеристики (неизменямость, упорядоченность), сравнение со списками • Списочные выражения с if • Задачи
7	2 по 45 минут	Методы строк. Методы split и join. Введение в списочные выражения. План: • Понятие метода, отличие от функции. • Список методов строк с краткими пояснениями • Функции dir и help, использование документации • Методы split и join • Списочные выражения (без if) • Задачи
8	2 по 45 минут	Знакомство со словарями. План: • Понятие словаря, характеристики (изменяемость, неупорядоченность) • Создание словаря • Обращение к элементам словаря • Добавление и удаление, проверка наличия элемента в словаре • Допустимые типы ключей • Перебор элементов словаря • Методы словарей (кратко) • Задачи
9	2 по 45 минут	Функции. Возвращение значений из функций. План: • Понятие функции, проблемы, решаемые функциями (группировка кода, повторное использование) • Синтаксис определения функций, синтаксис вызова • Введение в аргументы функции, понятие сигнатуры • Возвращение значений, return • Возврат нескольких значений • Задачи
10	2 по 45 минут	Файлы. • Принципы хранения файлов в памяти компьютера • Стандартный поток ввода • Открытие и закрытие файла

		• Чтение файла в текстовом режиме • Менеджер контекста • Запись в файл в текстовом режиме • Режимы открытия файлов • Стандартный поток вывода • Задачи

Разбор заданий демонстрационного варианта.

Варианты к конкурсу представлены в виде кейсов – крупных задач, содержащих объединенные по замыслу подзадачи. Решение каждого такого кейса нужно представить в виде набора функций, решающих подзадачи. В случае демонстрационного варианта нужно реализовать некое подобие системы управления базой данных (СУБД) для хранения и обработки информации об учениках некоторого учебного заведения. Задача разбивается на шесть подзадач, каждая из которых выражена в виде требования реализовать определенную функциональность воплощаемой «СУБД»: добавление записи, вывод записи, удаление записи, вывод всех данных в сортированном виде, сохранение данных и загрузка данных. Решение каждой подзадачи необходимо оформить в виде функции. Функция может (при необходимости) возвращать значения, печатать их в консоль или вовсе работать взаимодействовать с остальной программой через параметры (аргументы).

Решение демонстрационного варианта, распространяемое вместе с условием, содержит подробные комментарии с описанием реализации каждой подзадачи, но стоит обратить внимание дополнительно на несколько деталей. Основная часть программы, в которой производится считывание и разбор команд с консоли, оформлена в виде функции main. Это сделано для удобства и не требуется в варианте. В решении также используется несколько приемов: хранение данных об учениках в словаре для ускорения поиска, организация ветвления также с помощью словаря в основной части программы. Можно обратить на это внимание учащихся при разборе.

Ниже прилагается решение с комментариями:

```
# функция для добавления ученика
# data (здесь и далее) - словарь, в котором хранится вся
информация об учениках, ключи - имена
def add_pupil(name, birth_year, cs_score, math_score,
physics_score, data):
    data[name] = (name, birth_year, cs_score, math_score,
physics_score)  # доавляем в словарь кортеж с информацией


# функция для вывода информации об ученике
def show_pupil(name, data):
    print(*data[name])  # выводим информацию об ученике через
пробел, так как sep=' ' по умолчанию


# функция для удаления ученика
def del_pupil(name, data):
```

```
    del data[name]  # удаляем ученика с именем name


# функция для вывода информации об ученике
def show_summary(sort_field, data):
    # сортируем кортежи с информацией об учениках по элементу с
заданным номером
    sorted_rows = sorted(data.values(), key=lambda x:
x[int(sort_field)])
    for row in sorted_rows:  # проходим по сортированным ученикам
...
        print(*row)  # ... и выводим их


# функция для сохранения информации об учениках в файл
def save(file_path, data):
    # открываем файл в режиме записи с кодировкой UTF-8 для
корректной записи кириллицы
    with open(file_path, mode='w', encoding='utf-8') as
output_file:
        for row in data.values():  # проходим по значениям
(кортежам) в словаре с информацией об учениках ...
            print(*row, file=output_file)  # ... и выводим их в
открытый файл


# функция для чтения информации об учениках из файла
def load(file_path, data):
    data.clear()  # сбрасываем текущую информацию об учениках,
чтобы "с нуля" загрузить информацию из файла
    # открываем файл в режиме чтения с кодировкой UTF-8 для
корректного чтения кириллицы
    with open(file_path, mode='r', encoding='utf-8') as
input_file:
        for line in input_file:  # читаем файл построчно
            line_parts = line.rstrip('\n').split()  # срезаем
символ переноса строки с конца и делим строку по пробельным
символам
            add_pupil(*line_parts, data)  # добавляем считанного
ученика в словарь data


# главная функция
def main():
    data = dict()


    # составляем словарь соответствия названий команд с функциями
(функциональный подход)
    # альтернатива - вызов функций в блоках if-else
    cmd2func = {'ДОБАВИТЬ': add_pupil,
```

```
                'ПОКАЗАТЬ': show_pupil,
                'УДАЛИТЬ': del_pupil,
                'СВОДКА': show_summary,
                'СОХРАНИТЬ': save,
                'ЗАГРУЗИТЬ': load}


    # запускаем цикл чтения команд
    while True:
        cmd, *cmd_args = input().split()   # считываем команду,
отделяя название от аргументов
        cmd2func[cmd](*cmd_args, data)   # вызываем соотвествующую
функцию-обработчик


main()   # вызов главной функции
```

Указания для учителей по подготовке к конкурсу.

Целью курса является формирование у обучающегося знаний и умений, необходимых для успешного дальнейшего развития и деятельности в области программирования. Для достижения поставленной цели обучение по программе призвано обеспечить решение следующих задач:

- формирование в процессе решения практических задач у учащихся навыков применения методов программирования на языке Python;
- овладение приемами написания программ на языке Python;
- осознание практической применимости выполняемых учебных задач программирования в будущей профессиональной деятельности.

При подготовке особое внимание стоит уделить работе с файлами. Указать на то, что для открытия/закрытия файлов лучше использовать менеджер контекстов (with ... as …). Открывать файлы лучше в кодировке UTF-8, что решит проблемы с чтением и записью кириллицы. Для вывода в файл, открытый в режиме записи, лучше использовать уже привычную ученикам функцию print, передавая в нее именованный аргумент file. Также стоит обратить внимание на использование функций. Ученики часто забывают, что если функция возвращает значение (используется return), то это значение нужно затем в основной программе как-то использовать (например, вывести в консоль) и наоборот, если функция уже печатает значение в консоль, то дополнительно ее возвращаемое значение (чаще всего None) выводить не нужно. Также стоит напомнить о принципах сортировки коллекций. Не нужно реализовывать сортировку самостоятельно, лучше воспользоваться встроенными функциями и методами. Это приводит к увеличению производительности и снижению числа ошибок. Часто ученики путают функцию sorted с методом sort у списков или вовсе не знают разницы. Функцию sorted можно использовать для получения сортированного списка с элементами из любой коллекции (итерируемого объекта), а метод sort есть только у списков (из основных коллекций). Sort меняет список, на котором вызывается, а sorted возвращает новый –

сортированный. Очень желательно также разобрать сортировку при передаче ключа сравнения key. Это может избавить учащихся от «изобретения» своих циклов для сортировки, что приведет к экономии времени. Также можно обратить внимание на основные методы строк, в которых часто путаются ученики. Основными для них при решении конкурса будут split и join. Также нужно обратить внимание на особенности чтения строк из файлов: на конце строки остается символ переноса строки «\n», который необходимо обрезать методом rstrip или другим способом.

Программное обеспечение для обучения и выполнения задания:

1. ОС Windows 10
2. Python 3.8.0 или новее.
3. JetBrains PyCharm Community Edition последней версии.

Список учебной и методической литературы:

1. Бэрри П. Изучаем программирование на Python. — Эксмо, 2017. — 624 с.
2. Лутц М. Изучаем Python. 4-е издание. — Символ-Плюс, 2019. — 1280 с.

Методические материалы к кейсу по курсу «Основы программирования на C++»