Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебный предмет
«Конструирование программ и языки программирования»

Инструкция

по выполнению лабораторной работы №12
«Разработка программ с использованием структур и перечислений»

Минск 2024 г.

Лабораторная работа № 12

Тема работы: «Разработка программ с использованием структур и перечислений»

1 Цель работы

Сформировать умения разрабатывать алгоритмы и реализовать программы с применением переменных структур и перечислений.

2 Задание

Номер варианта соответствует номеру по списку в журнале.

1. Для получения места в общежитии формируется список студентов, который включает Ф.И.О. студента, группу, средний балл, доход на члена семьи. Общежитие в первую очередь предоставляется тем, у кого доход на члена семьи меньше двух минимальных зарплат, затем остальным в порядке уменьшения среднего балла. Вывести список очередности предоставления мест в общежитии.
2. Описать структуру с именем STUDENT, содержащую следующие поля: - фамилия и инициалы; - номер группы; - успеваемость (массив из пяти элементов). Написать программу, выполняющую следующие действия: - ввод с клавиатуры данных; записи должны быть упорядочены по алфавиту.
3. Создать перечисление должностей Vacancies {Manager, Boss, Clerk, Salesman, etc.}. Создать структуру «Employee», состоящую из:
   - поля name строкового типа;
   - поля vacancy типа Vacancies;
   - поля зарплата целого типа;
   - поля дата приема на работу типа int[3].

   Создать массив сотрудников. Длина массива задается пользователем, заполнение массива производится им же. Вывести полную информацию обо всех сотрудниках.
4. Описать структуру с именем STUDENT, содержащую следующие поля: - фамилия и инициалы; - номер группы; - успеваемость (массив из пяти элементов). Написать программу, выполняющую следующие действия: - ввод с клавиатуры данных; - вывод на экран фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2; если таких студентов нет, вывести соответствующее сообщение.
5. Описать структуру для групп студентов. Поля структуры – произвольные. Необходимо вывести информацию о студентах, чьи фамилии начинаются на I, F. Данные вводятся с клавиатуры.
6. Используя перечисление определить количество дней в указанном месяце.
7. В программе заданы месяц и год двух дат. Пользователь вводит еще одну дату (только месяц и год). Определить, принадлежит ли третья дата диапазону от первой даты до второй включительно. Задачу решить с использованием структуры данных.

8. Пользователь вводит данные о количестве студентов, их фамилии, имена и балл для каждого. Программа должна определить средний балл и вывести фамилии и имена студентов, чей балл выше среднего.

9. Написать программу, в которой хранятся данные о товарах, их количестве и цене. При запуске программы эта информация выводится на экран. Далее пользователю должно предлагаться вводить номера товаров и их новое количество. После этого все данные о товарах должны снова выводиться на экран. Задачу решить с использованием структуры данных.

10. Используя структуры, написать программу сложения и умножения двух комплексных чисел.

11. Напишите программу, которая сначала по первой букве должности, введенной пользователем, определяет соответствующее значение переменной, помещает это значение в переменную, а затем выводит полностью название должности, первую букву которой ввел пользователь.

12. Пользователь вводит название учебной дисциплины. Вывести всех преподавателей, которые ведут данную дисциплину.

13. Создать структуру и перечисление. Вывести сведения о машинах, прошедших техосмотра менее года назад.

14. Организовать поиск в структуре по типу самолёта, типы самолёта уже введены в программе.

15. Пользователь вводит название поры года. Необходимо вывести все названия месяцев, которые принадлежат данной поре года.

3 Оснащение работы

ПК, среда Visual Studio 2019, MSword.

4 Основные теоретические сведения

Как вам должно быть уже известно, классы относятся к ссылочным типам данных. Это означает, что объекты конкретного класса доступны по ссылке, в отличие от значе ний простых типов, доступных непосредственно. Но иногда прямой доступ к объектам как к значениям простых типов оказывается полезно иметь, например, ради повыше ния эффективности программы. Ведь каждый доступ к объектам (даже самым мелким) по ссылке связан с дополнительными издержками на расход вычислительных ресурсов и оперативной памяти. Для разрешения подобных затруднений в C# предусмотрена структура, которая подобна классу, но относится к типу значения, а не к ссылочному типу данных.

Структуры объявляются с помощью ключевого слова struct и с точки зрения син таксиса подобны классам. Ниже приведена общая форма объявления структуры:

```
struct имя : интерфейсы {
   // объявления членов
}
```

где имя обозначает конкретное имя структуры.

Одни структуры не могут наследовать другие структуры и классы или служить в качестве базовых для других структур и классов. (Разумеется, структуры, как и все остальные типы данных в C#, наследуют класс object.) Тем не менее в структуре мож но реализовать один или несколько интерфейсов, которые указываются после имени структуры списком через запятую. Как и у классов, у каждой структуры имеются свои члены: методы, поля, индексаторы, свойства, операторные методы и события. В струк турах допускается также определять конструкторы, но не деструкторы. В то же время для структуры нельзя определить конструктор, используемый по умолчанию (т.е. кон структор без параметров). Дело в том, что конструктор, вызываемый по умолчанию, определяется для всех структур автоматически и не подлежит изменению. Такой кон структор инициализирует поля структуры значениями, задаваемыми по умолчанию. А поскольку структуры не поддерживают наследование, то их члены нельзя указывать как abstract, virtual или protected.

Объект структуры может быть создан с помощью оператора new таким же обра зом, как и объект класса, но в этом нет особой необходимости. Ведь когда используется оператор new, то вызывается конструктор, используемый по умолчанию. А когда этот оператор не используется, объект по-прежнему создается, хотя и не инициализируется.

В этом случае инициализацию любых членов структуры придется выполнить вручную. В приведенном ниже примере программы демонстрируется применение структу ры для хранения информации о книге.

```
// Продемонстрировать применение структуры.
using System;
// Определить структуру.
struct Book {
   public string Author;
   public string Title;
   public int Copyright;

   public Book(string a, string t, int c) {
      Author = a;
      Title = t;
      Copyright = c;
   }
}
// Продемонстрировать применение структуры Book.
class StructDemo {
   static void Main() {
      Book book1 = new Book("Герберт Шилдт",
                            "Полный справочник по C# 4.0",
                            2010) ; // вызов явно заданного конструктора
      Book book2 = new Book(); // вызов конструктора по умолчанию
      Book book3; // конструктор не вызывается
```

```
        Console.WriteLine(book1.Author + ", " +
                    book1.Title + ", (c) " + book1.Copyright);
        Console.WriteLine();
        if(book2.Title == null)
          Console.WriteLine("Член book2.Title пуст.");
        // А теперь ввести информацию в структуру book2.
        book2.Title = "О дивный новый мир";
        book2.Author = "Олдос Хаксли";
        book2.Copyright = 1932;
        Console.Write("Структура book2 теперь содержит:\n");
        Console.WriteLine(book2.Author + ", " +
                    book2.Title + ", (c) " + book2.Copyright);
        Console.WriteLine();
        // Console.WriteLine(book3.Title); // неверно, этот член структуры
                                  // нужно сначала инициализировать
        book3.Title = "Красный шторм";

        Console.WriteLine(book3.Title); // теперь верно
    }
}
```

При выполнении этой программы получается следующий результат.

```
Герберт Шилдт, Полный справочник по C# 4.0, (c) 2010

Член book2.Title пуст.
Структура book2 теперь содержит:
Олдос Хаксли, О, дивный новый мир, (c) 1932

Красный шторм
```

Как демонстрирует приведенный выше пример программы, структура может быть инициализирована с помощью оператора new для вызова конструктора или же путем простого объявления объекта. Так, если используется оператор new, то поля структуры инициализируются конструктором, вызываемым по умолчанию (в этом случае во всех полях устанавливается задаваемое по умолчанию значение), или же конструктором, определяемым пользователем. А если оператор new не используется, как это имеет место для структуры book3, то объект структуры не инициализируется, а его поля должны быть установлены вручную перед тем, как пользоваться данным объектом.

Когда одна структура присваивается другой, создается копия ее объекта. В этом за ключается одно из главных отличий структуры от класса. Как пояснялось ранее в этой книге, когда ссылка на один класс присваивается ссылке на другой класс, в итоге ссылка в левой части оператора присваивания указывает на тот же самый объект, что и ссылка в правой его части. А когда переменная одной структуры присваивается переменной другой структуры,

создается копия объекта структуры из правой части оператора при сваивания. Рассмотрим в качестве примера следующую программу.

```
// Скопировать структуру.
using System;
// Определить структуру.
struct MyStruct {
   public int x;
}
// Продемонстрировать присваивание структуры.
class StructAssignment {
   static void Main() {
      MyStruct a;
      MyStruct b;
      a.x = 10;
      b.x = 20;
      Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x);
      a = b;
      b.x = 30;
      Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x);
   }
}
```

Вот к какому результату приводит выполнение этой программы.

```
a.x 10, b.x 20
a.x 20, b.x 30
```

Как показывает приведенный выше результат, после присваивания

```
a = b;
```

переменные структуры a и b по-прежнему остаются совершенно обособленными, т.е. переменная a не указывает на переменную b и никак не связана с ней, помимо того, что она содержит копию значения переменной b. Ситуация была бы совсем иной, если бы переменные a и b были ссылочного типа, указывая на объекты определенного клас са. В качестве примера ниже приведен вариант предыдущей программы, где демонстрируется присваивание переменных ссылки на объекты определенного класса.

```
// Использовать ссылки на объекты определенного класса.
using System;
// Создать класс.
class MyClass {
   public int x;
}
// Показать присваивание разных объектов данного класса.
class ClassAssignment {
   static void Main() {
      MyClass a = new MyClass();
      MyClass b = new MyClass();
      a.x = 10;
```

```
        b.x = 20;
        Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x);
        a = b;
        b.x = 30;
        Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x);
    }
}
```

Выполнение этой программы приводит к следующему результату.

```
a.x 10, b.x 20
a.x 30, b.x 30
```

Как видите, после того как переменная b будет присвоена переменной a, обе пере менные станут указывать на один и тот же объект, т.е. на тот объект, на который перво начально указывала переменная b.

О назначении структур

В связи с изложенным выше возникает резонный вопрос: зачем в C# включена структура, если она обладает более скромными возможностями, чем класс? Ответ на этот вопрос заключается в повышении эффективности и производительности про грамм. Структуры относятся к типам значений, и поэтому ими можно оперировать непосредственно, а не по ссылке. Следовательно, для работы со структурой вообще не требуется переменная ссылочного типа, а это означает в ряде случаев существенную экономию оперативной памяти. Более того, работа со структурой не приводит к ухуд шению производительности, столь характерному для обращения к объекту класса. Ведь доступ к структуре осуществляется непосредственно, а к объектам — по ссылке, поскольку классы относятся к данным ссылочного типа. Косвенный характер доступа к объектам подразумевает дополнительные издержки вычислительных ресурсов на каждый такой доступ, тогда как обращение к структурам не влечет за собой подобные издержки. И вообще, если нужно просто сохранить группу связанных вместе данных, не требующих наследования и обращения по ссылке, то с точки зрения производи тельности для них лучше выбрать структуру.

Ниже приведен еще один пример, демонстрирующий применение структуры на практике. В этом примере из области электронной коммерции имитируется запись транзакции. Каждая такая транзакция включает в себя заголовок пакета, содержащий номер и длину пакета. После заголовка следует номер счета и сумма транзакции. Заго ловок пакета представляет собой самостоятельную единицу информации, и поэтому он организуется в отдельную структуру, которая затем используется для создания за писи транзакции или же информационного пакета любого другого типа.

```
// Структуры удобны для группирования небольших объемов данных.
using System;
// Определить структуру пакета.
struct PacketHeader {
    public uint PackNum; // номер пакета
    public ushort PackLen; // длина пакета
```

```
}
// Использовать структуру PacketHeader для создания записи транзакции
// в сфере электронной коммерции.
class Transaction {
   static uint transacNum = 0;
   PacketHeader ph; // ввести структуру PacketHeader в класс Transaction
   string accountNum;
   double amount;
   public Transaction(string acc, double val) {
      // создать заголовок пакета
      ph.PackNum = transacNum++;
      ph.PackLen = 512; // произвольная длина
      accountNum = acc;
      amount = val;
   }
   // Сымитировать транзакцию.
   public void sendTransaction() {
      Console.WriteLine("Пакет #: " + ph.PackNum +
                        ", Длина: " + ph.PackLen +
                        ",\n Счет #: " + accountNum +
                        ", Сумма: {0:C}\n", amount);
   }
}
// Продемонстрировать применение структуры в виде пакета транзакции.
class PacketDemo {
   static void Main() {
      Transaction t = new Transaction("31243", -100.12);
      Transaction t2 = new Transaction("AB4655", 345.25);
      Transaction t3 = new Transaction("8475-09", 9800.00);

      t.sendTransaction();
      t2.sendTransaction();
      t3.sendTransaction();
   }
}
```

Вот к какому результату может привести выполнение этого кода.

```
Пакет #: 0, Длина: 512,
   Счет #: 31243, Сумма: ($100.12)

Пакет #: 1, Длина: 512,
   Счет #: AB4655, Сумма: $345.25
Пакет #: 2, Длина: 512,
   Счет #: 8475-09, Сумма: $9,800.00
```

Структура PacketHeader оказывается вполне пригодной для формирования заго ловка пакета транзакции, поскольку в ней хранится очень

небольшое количество дан ных, не используется наследование и даже не содержатся методы. Кроме того, работа со структурой PacketHeader не влечет за собой никаких дополнительных издержек, связанных со ссылками на объекты, что весьма характерно для класса. Следовательно, структуру PacketHeader можно использовать для записи любой транзакции, не сни жая эффективность данного процесса.

Любопытно, что в C++ также имеются структуры и используется ключевое слово struct. Но эти структуры отличаются от тех, что имеются в C#. Так, в C++ структура относится к типу класса, а значит, структура и класс в этом языке практически равно ценны и отличаются друг от друга лишь доступом по умолчанию к их членам, которые оказываются закрытыми для класса и открытыми для структуры. А в C# структура от носится к типу значения, тогда как класс — к ссылочному типу.

Перечисления

Перечисление представляет собой множество именованных целочисленных констант. Перечислимый тип данных объявляется с помощью ключевого слова enum. Ниже при ведена общая форма объявления перечисления:

enum имя {список_перечисления};

где имя — это имя типа перечисления, а список_перечисления — список идентифи каторов, разделяемый запятыми. В приведенном ниже примере объявляется перечисление Apple различных сортов яблок.

```
enum Apple { Jonathan, GoldenDel, RedDel, Winesap,
             Cortland, McIntosh };
```

Следует особо подчеркнуть, что каждая символически обозначаемая константа в перечислении имеет целое значение. Тем не менее неявные преобразования пере числимого типа во встроенные целочисленные типы и обратно в C# не определены, а значит, в подобных случаях требуется явное приведение типов. Кроме того, приве дение типов требуется при преобразовании двух перечислимых типов. Но поскольку перечисления обозначают целые значения, то их можно, например, использовать для управления оператором выбора switch или же оператором цикла for.

Для каждой последующей символически обозначаемой константы в перечислении задается целое значение, которое на единицу больше, чем у предыдущей константы. По умолчанию значение первой символически обозначаемой константы в перечисле нии равно нулю. Следовательно, в приведенном выше примере перечисления Apple константа Jonathan равна нулю, константа GoldenDel — 1, константа RedDel — 2 и т.д.

Доступ к членам перечисления осуществляется по имени их типа, после которого следует оператор-точка. Например, при выполнении фрагмента кода

```
Console.WriteLine(Apple.RedDel + " имеет значение " +
                  (int)Apple.RedDel);
```

выводится следующий результат.

```
RedDel имеет значение 2
```

Как показывает результат выполнения приведенного выше фрагмента кода, для вы вода перечислимого значения используется его имя. Но для получения этого значения требуется предварительно привести его к типу int.

Ниже приведен пример программы, демонстрирующий применение перечисле ния Apple.

```
// Продемонстрировать применение перечисления.
using System;
class EnumDemo {
   enum Apple { Jonathan, GoldenDel, RedDel, Winesap,
   Cortland, McIntosh };
   static void Main() {
      string[] color = {
         "красный",
         "желтый",
         "красный",
         "красный",
         "красный",
         "красновато-зеленый"
      };
      Apple i; // объявить переменную перечислимого типа
      // Использовать переменную i для циклического
      // обращения к членам перечисления.
      for(i = Apple.Jonathan; i <= Apple.McIntosh; i++)
         Console.WriteLine(i + " имеет значение " + (int)i);
      Console.WriteLine();

      // Использовать перечисление для индексирования массива.
      for(i = Apple.Jonathan; i <= Apple.McIntosh; i++)
         Console.WriteLine("Цвет сорта " + i + " — " +
      color[(int)i]);
   }
}
```

Ниже приведен результат выполнения этой программы.

```
Jonathan имеет значение 0
GoldenDel имеет значение 1
RedDel имеет значение 2
Winsap имеет значение 3
Cortland имеет значение 4
McIntosh имеет значение 5
Цвет сорта Jonathan - красный
Цвет сорта GoldenDel - желтый
Цвет сорта RedDel - красный
Цвет сорта Winsap - красный
Цвет сорта Cortland - красный
Цвет сорта McIntosh - красновато-зеленый
```

Обратите внимание на то, как переменная типа Apple управляет циклами for. Значения символически обозначаемых констант в перечислении Apple начинаются с нуля, поэтому их можно использовать для индексирования массива, чтобы получить цвет каждого сорта яблок. Обратите также внимание на необходимость производить приведение типов, когда перечислимое значение используется для индексирования массива. Как упоминалось выше, в C# не предусмотрены неявные преобразования перечислимых типов в целочисленные и обратно, поэтому для этой цели требуется явное приведение типов.

И еще одно замечание: все перечисления неявно наследуют от класса System.Enum, который наследует от класса System.ValueType, а тот, в свою очередь, — от класса object.

Инициализация перечисления

Значение одной или нескольких символически обозначаемых констант в перечис лении можно задать с помощью инициализатора. Для этого достаточно указать после символического обозначения отдельной константы знак равенства и целое значение. Каждой последующей константе присваивается значение, которое на единицу больше значения предыдущей инициализированной константы. Например, в приведенном ниже фрагменте кода константе RedDel присваивается значение 10.

```
enum Apple { Jonathan, GoldenDel, RedDel = 10, Winesap,
        Cortland, McIntosh };
```

Указание базового типа перечисления

По умолчанию в качестве базового для перечислений выбирается тип int, тем не менее перечисление может быть создано любого целочисленного типа, за исключени ем char. Для того чтобы указать другой тип, кроме int, достаточно поместить этот тип после имени перечисления, отделив его двоеточием. В качестве примера ниже за дается тип byte для перечисления Apple.

```
enum Apple : byte { Jonathan, GoldenDel, RedDel,
            Winesap, Cortland, McIntosh };
```

Теперь константа Apple.Winesap, например, имеет количественное значение типа byte.

Применение перечислений

На первый взгляд перечисления могут показаться любопытным, но не очень нуж ным элементом C#, но на самом деле это не так. Перечисления очень полезны, когда в программе требуется одна или несколько специальных символически обозначаемых констант. Допустим, что требуется написать программу для управления лентой кон вейера на фабрике. Для этой цели можно создать метод Conveyor(), принимающий в качестве параметров следующие команды: "старт", "стоп", "вперед" и "назад". Вместо того чтобы передавать методу Conveyor() целые значения, например, 1 — в качестве команды "старт", 2 — в качестве команды "стоп" и так далее, что чревато ошибками, можно создать перечисление, чтобы присвоить этим значениям

содержательные сим волические обозначения. Ниже приведен пример применения такого подхода.

```
// Сымитировать управление лентой конвейера.
using System;
class ConveyorControl {
  // Перечислить команды конвейера.
  public enum Action { Start, Stop, Forward, Reverse };

  public void Conveyor(Action com) {
    switch(com) {
      case Action.Start:
        Console.WriteLine("Запустить конвейер.");
        break;
      case Action.Stop:
        Console.WriteLine("Остановить конвейер.");
        break;
      case Action.Forward:
        Console.WriteLine("Переместить конвейер вперед.");
        break;
      case Action.Reverse:
        Console.WriteLine("Переместить конвейер назад.");
        break;
    }
  }
}
class ConveyorDemo {
  static void Main() {
    ConveyorControl c = new ConveyorControl();

    c.Conveyor(ConveyorControl.Action.Start);
    c.Conveyor(ConveyorControl.Action.Forward);
    c.Conveyor(ConveyorControl.Action.Reverse);
    c.Conveyor(ConveyorControl.Action.Stop);
  }
}
```

Вот к какому результату приводит выполнение этого кода.

```
Запустить конвейер.
Переместить конвейер вперед.
Переместить конвейер назад.
Остановить конвейер.
```

Метод Conveyor() принимает аргумент типа Action, и поэтому ему могут быть переданы только значения, определяемые в перечислении Action. Например, ниже приведена попытка передать методу Conveyor() значение 22.

```
c.Conveyor(22); // Ошибка!
```

Эта строка кода не будет скомпилирована, поскольку отсутствует предваритель но заданное преобразование типа int в перечислимый тип Action. Именно это и препятствует передаче неправильных команд методу Conveyor(). Конечно, такое преобразование можно организовать принудительно с помощью приведения типов, но это было бы преднамеренным, а не случайным или неумышленным действием. Кроме того, вероятность неумышленной передачи пользователем неправильных ко манд методу Conveyor() сводится с минимуму благодаря тому, что эти команды обо значены символическими именами в перечислении.

В приведенном выше примере обращает на себя внимание еще одно интересное обстоятельство: перечислимый тип используется для управления оператором switch. Как упоминалось выше, перечисления относятся к целочисленным типам данных, и поэтому их вполне допустимо использовать в операторе switch.

5. Порядок выполнения работы

1. Выделить ключевые моменты задачи.
2. Построить алгоритм и теоретическую объектную модель решения задачи.
3. Запрограммировать полученные алгоритмы и объектную модель.

6. Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____

Фамилия, инициалы учащегося _____

Дата выполнения работы _____

Тема работы: _____

Цель работы: _____

Оснащение работы: _____

Результат выполнения работы: _____

7. Контрольные вопросы и задания

1. Структура – это…?
2. Перечисление – это …?
3. В каких случаях лучше использовать перечисления, а в каких – структуры?

8. Рекомендуемая литература

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. СПб. : Изд-во Питер, 2021. 896 с.

2. Прайс, М. Дж. C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Дж. Прайс. СПб : Изд-во Питер, 2023. 848 с.

3. Васильев, А.Н. Программирование на C# для начинающих. Особенности языка / А.Н. Васильев. М. : Эксмо, 2022. 528 с.

4. Фримен, А. ASP.NET Core 3 с примерами на C# для профессионалов / А. Фримен. СПб. : Изд-во Вильямс, 2021. 1184 с.