22.12.20г\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

19ИСиП 1Д \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Основы алгоритмизации и программирования

ТЕМА: Составление программы

Разработку программ будем вести в интегрированной среде разработки SharpDevelop. Это система разработки с открытым кодом, внешне очень похожа на Microsoft Visual Studio. SharpDevelop позволяет выполнять всё то же, что вы делали бы в Visual Studio.

*Integrated Development Environment* (интегрированная среда разработки), или, сокращенно, IDE – это программный продукт, объединяющий текстовый редактор, компилятор, отладчик и справочную систему.

Любая программа, создаваемая в среде SharpDevelop, даже такая простая, как "Hello World!", всегда оформляется как отдельный *проект* (*project*). Проект — это набор взаимосвязанных исходных файлов и, возможно, дополнительных файлов, компиляция и компоновка которых позволяет создать исполняемую программу. Однако разработчики и SharpDevelop, и Visual Studio пошли еще дальше, стремясь удовлетворить потребности не только программистов-одиночек, но и больших коллективов разработчиков программных продуктов. Так появилось понятие *решения* (*solution*). Решение может содержать любое количество различных проектов, сгруппированных вместе для согласованной разработки: от отдельного приложения до библиотеки классов или целого программного пакета. Очевидно, что для решения наших учебных задач каждая программа будет воплощаться в виде одного проекта, поэтому *решение* у нас всегда будет содержать ровно один проект.

Среда SharpDevelop позволяет разрабатывать программы на языках C#, Basic .NET, а также на ряде других языков программирования (в зависимости от версии SharpDevelop). В настоящее время обычно используют версии SharpDevelop 2.2 или SharpDevelop 3.2.

Существует целый ряд заготовок (шаблонов) готовых проектов, на базе которых можно разрабатывать консольные, оконные приложения и создавать модули компонентов.

***Порядок создания консольного приложения***

Изучение возможностей языка программирования лучше всего начинать с разработки консольных приложений. Порядок создания таких программ следующий:

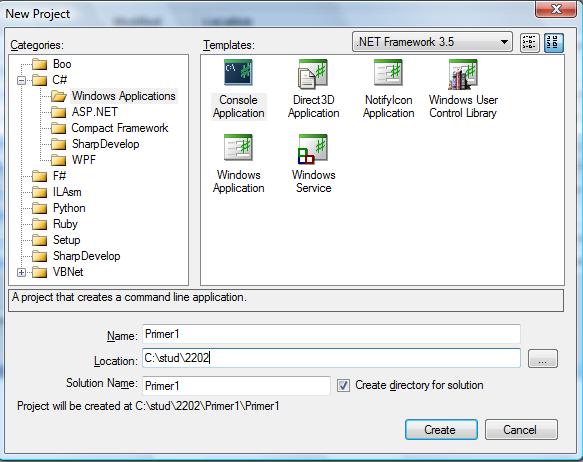
1)Запускаем SharpDevelop из главного меню:

**Пуск/Программы/SharpDevelop 2.2/SharpDevelop 2.2**

или щелкаем по ярлыку программы SharpDevelop 2.2 на рабочем столе.

2)На стартовой странице (**Start page**), если она открыта, нажимаем на кнопку **New solution** или в меню программы выбираем **File/New/Solution...**

3)Откроется окно создания нового проекта:



Необходимо выбрать тип проекта: «**Console Application**», задать имя проекта, например: Primer1, определить папку, в которой будет сохранён проект (**Location**), например, 2202 (номер группы).

Затем нажимаем на кнопку **Create** (Создать). Всё, проект создан. В окне редактирования будет виден текст минимальной программы на языке C#:

**/\***

**\* Created by SharpDevelop.**

**\* User: User**

**\* Date: 15.02.2011**

**\* Time: 19:50**

**\***

**\* To change this template use Tools | Options | Coding | Edit Standard Headers.**

**\*/**

**using System;**

**namespace Primer1**

**{**

**class Program**

**{**

**public static void Main(string[] args)**

**{**

**Console.WriteLine("Hello World!");**

**// TODO: Implement Functionality Here**

**Console.Write("Press any key to continue ...");**

**Console.ReadKey(true);**

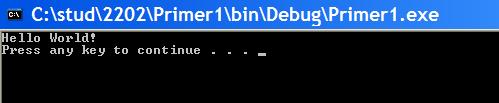
**}**

**}**

**}**

Запуск программы на выполнение осуществляется путём нажатия на кнопку (**!**) на панели инструментов или выбором пункта меню **Debug/Run without debugger**.

Если запустить на выполнение эту программу, то в окне выводы увидим следующее:



Теперь остаётся вместо области **TODO** написать тот текст, который необходим для решения поставленной задачи.

**Пояснения к этой минимальной программе**

1)В начале текста программы мы видим многострочный комментарий, добавленный системой. Он начинается парой символов **/\*** (косая черта и звёздочка) и заканчивается символами **\*/** (звёздочка и косая черта). Этот комментарий можно удалить или заменить на более подходящий.

2)Далее идёт строка

**using System;**

Данная строка указывает на то, что в программе будет использована область имён **System**. Эта область имён является обязательной для любой программы на языке C#. Кроме области имён **System** при необходимости могут быть использованы и другие области имён.

3)Строка

**namespace Primer1**

является заголовком Вашей собственной области имён (**namespace** — ключевое слово, означающее область имён, **Primer1** — название этой области). Название области имён создаётся системой SharpDevelop и совпадает с названием проекта. Сама область имён представляет собой блок и начинается с открывающей фигурной скобки **{** (это первая фигурная скобка, записанная сразу же за заголовком нашей области имён), а заканчивается соответствующей закрывающей скобкой **}** (находится в последней строке программы).

4)Внутри области имён **Primer1** находится определение класса **Program** (**class** — ключевое слово, с помощью которого определяется класс, **Program** — название класса):

**class Program**

Тело класса также представляет собой блок. В нашей области имён определён всего один класс. При необходимости в области имён могут быть определены и другие классы. В тексте программы на языке C# в области имён обязательно должен быть определён хотя бы один класс. Имя данному классу также назначено системой. Его можно при желании заменить на другое.

Классы — основные строительные блоки программ на языке C#. Их изучением мы займёмся позже, а пока используем заготовку класса **Program** как есть.

5)Внутри класса **Program** определён единственный метод **Main()**. Вот его заголовок:

**public static void Main(string[] args)**

Тело метода также представляет из себя блок. Метод **Main()** — это главный метод в любой программе на языке C#. Он является обязательным для каждой программы на C#. Именно с метода **Main()** начинается выполнение программы.

Метод — это часть текста программы, оформленная особым образом. В методах реализуются все необходимые действия программы. Определение методов всегда даётся внутри классов. Количество методов в классах может быть произвольными. Более подробно изучать разработку и использование методов будем позже. Пока будем использовать заготовку метода **Main()** как есть.

6)Строка

**Console.WriteLine("Hello World!");**

предназначена для вывода текстового сообщения в консольном приложении. Дословно эту строку можно перевести так: вызывается метод **WriteLine()**, который определён в классе **Console**, а тот в свою очередь находится в стандартной области имён **System**. Символьная строка **"Hello World!"** — параметр метода **WriteLine()** — это та информация, что и будет выведена на экране.

7)Строка

**// TODO: Implement Functionality Here**

является однострочным комментарием и предназначена для пояснения: именно здесь и пишите тот программный текст, который Вам нужен.

8)Строки в конце программы

**Console.Write("Press any key to continue ...");**

**Console.ReadKey(true);**

предназначены для задержки выполнения программы. Не удаляйте их! Дело в том, что если эти строки отсутствуют в конце текста консольного приложения, то после выполнения вычислений окно консоли сразу же закроется. Вы можете просто не успеть увидеть результаты работы программы. Если эти строки остались Вами нетронутыми, то Вы можете спокойно изучать результаты работы программы столько времени, сколько Вам потребуется.

Собственно, для задержки выполнения программы нужен только метод **ReadKey()** (ожидается нажатие на любую клавишу), но и печать подсказки тоже не лишняя. Именно по выводу на экране текста **"Press any key to continue …"** («Нажмите на любую клавишу для продолжения...») становится понятно, что программа выполнена до конца. Нажмём на любую клавишу и окно вывода будет закрыто.

***Пример простой программы***

*Пример*. Написать программу для вычисления гипотенузы прямоугольного треугольника при условии, что известны катеты этого треугольника.

Обозначим длины катетов как в математике: **a** и **b**, гипотенузу — как **c**.

Порядок действий будет такой:

1. задаём с клавиатуры входные данные — значения **a** и **b**;
2. вычисляем значение гипотенузы по формуле

http://victor192007.narod.ru/files/ris/ris3_cs03a.gif;

1. печатаем полученное значение **c**.

Для написания этой программы воспользуемся заготовкой консольного приложения, приведённой выше.

**Вот окончательный текст программы**:

**/\***

**\* Created by SharpDevelop.**

**\* User: User**

**\* Date: 15.02.2011**

**\* Time: 19:50**

**\***

**\* To change this template use Tools | Options | Coding | Edit Standard Headers.**

**\*/**

**using System;**

**namespace Primer1**

**{**

**class Program**

**{**

**public static void Main(string[] args)**

**{**

**double a, b, c;**

**Console.Write("a=");**

**a = double.Parse(Console.ReadLine());**

**Console.Write("b=");**

**b = double.Parse(Console.ReadLine());**

**c = Math.Sqrt(a\*a + b\*b);**

**Console.WriteLine("c={0}", c);**

**Console.Write("Press any key to continue . . . ");**

**Console.ReadKey(true);**

**}**

**}**

**}**

**Результаты выполнения программы:**

**a=3**

**b=4**

**c=5**

**Press any key to continue . . .**

**Пояснения к тексту программы**

Рассмотрим текст метода **Main()** (всё остальное мы разобрали ранее).

1)Для решения поставленной задачи нам требуется **3** переменные. В строке

**double a, b, c;**

выделена память для этих переменных. Тип переменных — **double** (вещественный с удвоенной точностью), начальное значение переменным не задаём, так как намерены задать входные данные **a** и **b** с клавиатуры.

2)Строка

**Console.Write("a=");**

предназначена для вывода текста-подсказки. Пользователь программы должен понимать, что от него требуется. В данном случае будет напечатано

**a=**

(мы собираемся задать значение переменной **a**).

3)В строке

**a = double.Parse(Console.ReadLine());**

выполняется присваивание переменной **a** значения, введённого с клавиатуры. Как это работает? Мы вводим с клавиатуры число, например: число **3**, как в данном примере, но воспринимается это число как символьная строка. Поэтому вначале и вызывается метод **ReadLine()** класса **Console**. Именно этот метод возвращает символьною строку, то есть символьное представление нужного нам числа. Затем символьная строка поступает в метод **Parse()** структуры **double** и выполняется преобразование строки в вещественное число типа **double**. Последний шаг — присваивание полученного результата переменной **a**.

Обратите внимание на следующее: тип данных **double** является не просто стандартным типом данных в языке C#, он является структурой, то есть имеет совсем не такую уж простую организацию. Это же относится и ко всем другим стандартным типам данного языка. А вот в старых языках программирования (C, Pascal, Basic) стандартные типы данных были действительно простыми типами.

Подведём промежуточный итог. Два строки

**Console.Write("a=");**

**a = double.Parse(Console.ReadLine());**

обеспечивают ввод значения переменной **a** с клавиатуры, при этом печатается и подсказка, поясняющая, что требуется от пользователя программы. После выполнения этих действий на экране будет выведено:

**a=3**

4)В последующих двух строках строках

**Console.Write("b=");**

**b = double.Parse(Console.ReadLine());**

выполняется ввод с подсказкой значения другой переменной — переменной **b**. Делается всё точно также, как и для переменной **a**.

5)В строке

**c = Math.Sqrt(a\*a + b\*b);**

выполняется вычисление гипотенузы **c**. Справа от знака присваивания **=** записан вызов метода **Sqrt()**, определённый в классе **Math**. Метод **Sqrt()** предназначен для вычисления квадратного корня из вещественного числа.

6)В строке

**Console.WriteLine("c={0}", c);**

осуществляется вывод результата работы — печатается текст

**c=5**

В символьной строке, являющейся первым параметром метода **WriteLine()**, имеется не только текст, но и формат, определяющий правила вывода переменной **c**. Здесь используется простейший вариант формата **{0}**, который определяет порядковый номер выводимых параметров. Нумерация параметров всегда начинается с **0**. Именно этот номер и соответствует переменной **c**.

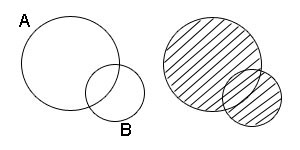
**Операция объединения множеств**

**Объединением множеств** *А* и *В* называется множество, обозначаемое https://function-x.ru/sets/s020.gif, состоящее из всех тех и только тех элементов, которые принадлежат или *А* или *В*, то есть

https://function-x.ru/sets/s021.gif.

Например, если https://function-x.ru/sets/s022.gif, https://function-x.ru/sets/s023.gif, https://function-x.ru/sets/s024.gif, то https://function-x.ru/sets/s025.gif, https://function-x.ru/sets/s026.gif, https://function-x.ru/sets/s027.gif.

Операции над множествами удобно иллюстрировать фигурами, называемыми диаграммами Венна (другое название - круги Эйлера). На рисунке ниже слева большим и малым кругами обозначены соответственно множества А и В, а справа - результат объединения этих множеств (заштрихованная фигура).



На основе теории множеств создана концепция реляционных баз данных, а на основе операций над множествами - [**реляционная алгебра и её операции**](https://function-x.ru/relation_algebra.html) - используемые в языке запросов к базам данных SQL. Операция объединения есть в реляционной алгебре.

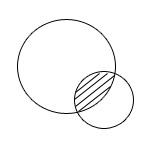
**Операция пересечения множеств**

**Пересечением множеств** *А* и *В* называется множество, обозначаемое https://function-x.ru/sets/s028.gif и состоящее из всех тех и только тех элементов, которые принадлежат каждому из множеств *А* и *В*, то есть

https://function-x.ru/sets/s029.gif.

Например, если https://function-x.ru/sets/s022.gif, https://function-x.ru/sets/s023.gif, https://function-x.ru/sets/s024.gif, то https://function-x.ru/sets/s030.gif, https://function-x.ru/sets/s031.gif, https://function-x.ru/sets/s032.gif.

На рисунке ниже - результат пересечения множеств А и В - заштрихованная фигура.



В одном из материалов сайта показано, как выглядит [**пересечение множеств решений систем линейных неравенств**](https://function-x.ru/systems_inequalities.html).

Операция пересечения есть в [**реляционной алгебре**](https://function-x.ru/relation_algebra.html), используемой для манипулирования данными в языках запросов к базам данных, например, SQL.

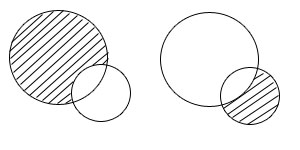
**Разность множеств**

**Разностью множеств** *А* и *В* называется множество, обозначаемое https://function-x.ru/sets/s033.gif и состоящее из всех тех и только тех элементов множества *А*, которые не являются элементами множества *В*, то есть

https://function-x.ru/sets/s034.gif.

Например, если https://function-x.ru/sets/s022.gif, https://function-x.ru/sets/s023.gif, https://function-x.ru/sets/s024.gif, то https://function-x.ru/sets/s035.gif, https://function-x.ru/sets/s036.gif, https://function-x.ru/sets/s037.gif, https://function-x.ru/sets/s038.gif, https://function-x.ru/sets/s039.gif.

На рисунке ниже слева - результат разности множеств А и В, а справа - результат разности множеств В и А.



Операция разности есть в [**реляционной алгебре**](https://function-x.ru/relation_algebra.html), используемой для манипулирования данными в языках запросов к базам данных, например, SQL.

Объекты, составляющие множества - объекты нашей интуиции или интеллекта - могут быть самой различной природы. В примере в первом параграфе мы разобрали множества, включающие набор продуктов. Множества могут состоять, например, и из всех букв русского алфавита. В математике изучаются множества чисел, например, состоящие из всех:

- натуральных чисел 0, 1, 2, 3, 4, ...

- простых чисел

- чётных целых чисел

и т.п. (основные числовые множества рассмотрены в [соответствующем параграфе](https://function-x.ru/sets1.html#paragraph3) этого материала).

Объекты, составляющие множество, называются его элементами. Можно сказать, что множество - это "мешок с элементами". Очень важно: в множестве не бывает одинаковых элементов.

Множества бывают конечными и бесконечными. Конечное множество - это множество, для которого существует натуральное число, являющееся числом его элементов. Например, множество первых пяти неотрицательных целых нечётных чисел https://function-x.ru/sets/s003.gif является конечным множеством. Множество, не являющееся конечным, называется бесконечным. Например, множество всех натуральных чисел является бесконечным множеством.

Если M - множество, а a - его элемент, то пишут: a∈M, что означает "a принадлежит множеству M".

Из первого (нулевого) примера на Паскале с продуктами, которые есть в тех или иных магазинах:

hleb∈**VETEROK**,

что означает: элемент "hleb" принадлежит множеству продуктов, которые есть в магазине "VETEROK".

Существуют два основных способа задания множеств: перечисление и описание.

Множество можно задать, перечислив все его элементы, например:

**VETEROK** = {hleb, syr, maslo},

**A** = {7, 14, 28}.

Перечислением можно задать только конечное множество. Хотя можно сделать это и описанием. Но бесконечные множества можно задать только описанием.

Для описания множеств используется следующий способ. Пусть p(x) - некоторое высказывание, которое описывает свойства переменной x, областью значений которых является множество **M**. Тогда через **M** = {x | p(x)} обозначаентся множество, состоящее из всех тех и только тех элементов, для которых высказывание p(x) истинно. Это выражение читается так: "Множество **M**, состоящее из всех таких x, что p(x)".

Например, запись

**M** = {x | x² - 3x + 2 = 0}

означает множество [**корней уравнения**](https://function-x.ru/sq_equations.html) x² - 3x + 2 = 0, т. е. множество {1, 2}. Это конечное множество.

А следующим описанием задаётся множество всех целых чисел больше 5:

**M** = {x∈**Z** | x > 5},

это множество является бесконечным.

Описанием предпочтительно задавать и конечные множества, в которых очень много элементов, например, множество всех натуральных чисел от 2 до 22³:

**M** = {x∈**N** | 2< x < 22³}.

Множество, не содержащее ни одного элемента, называется пустым и обозначается знаком ∅.

Множество может состоять из одного элемента. Необходимо различать элемент a и множество {a}, содержащее только один элемент a, хотя бы потому, что допускаются множества, элементы которых сами являются множествами. Например, множество a={2, 1} состоит из двух элементов 2 и 1, а множество {a}, состоит из одного элемента a, который сам является двухэлементным множеством.

Два множества называюся равными, если они состоят из одних и тех же элементов. Например, равны множество равносторонних треугольников и множество равноугольных треугольников, так как это одни и те же треугольники: если в треугольнике все стороны равны, то равны и все его углы. Обратно, из равенства всех трёх углов треугольника вытекает равенство всех трёх его сторон. Равны любые два конечных множетсва, отличающиеся друг от друга только лишь порядком их элементов, например, {a, b, c} = {c, a, b}.

Преподователь \_\_\_\_\_\_\_\_\_\_\_\_\_\_Дузаев И.К.