

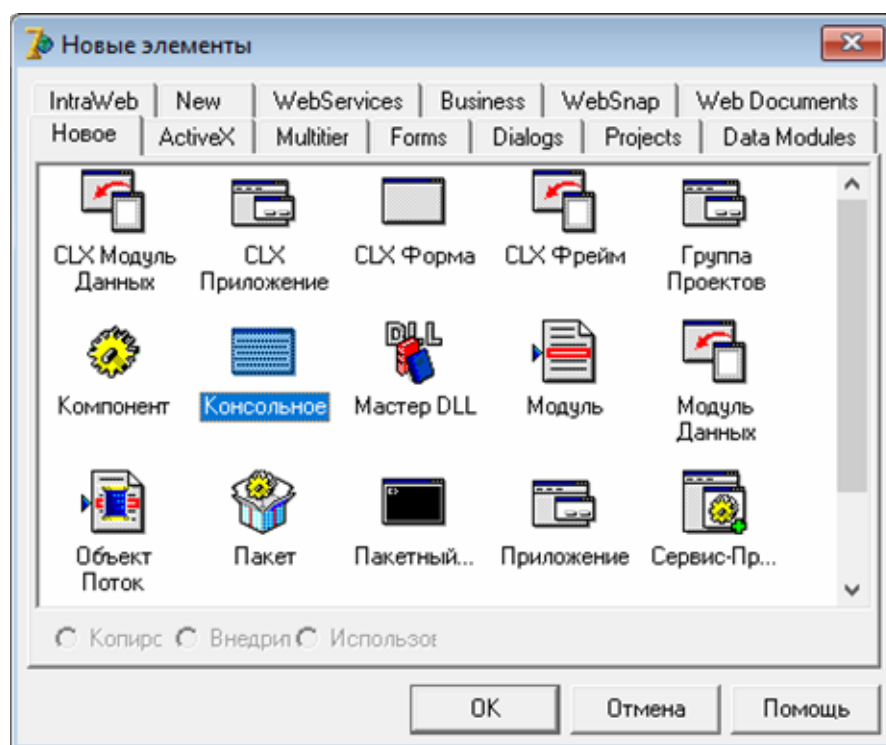
ТУБ №24:

Простейший ввод-вывод в Delphi

Простейшая программа

По уже сложившейся традиции первой программой, которую принято писать на любом языке программирования, является программа «Hello, world!» — программа, которая выводит пользователю приветственное сообщение и завершает работу.

Напишем такую программу. Поскольку задача выглядит не слишком сложной алгоритмически, пропустим такие этапы решения задачи, как выбор метода решения и структур данных, и сразу перейдём к написанию кода. Создадим для этого проект консольного приложения:



При этом среда создаст программный модуль примерно следующего содержания (Delphi 7):

```
program Project1;  
  
{$APPTYPE CONSOLE}  
  
uses  
    SysUtils;  
  
begin  
    { TODO -oUser -cConsole Main : Insert code here }  
end.
```

Модуль SysUtils, подключённый в предложении использования uses, нам не потребуется: будем использовать только возможности самого языка программирования. На месте TODO-комментария запишем одну единственную строку:

```

program Project1;

{$APPTYPE CONSOLE}

begin
    WriteLn('Hello, world!');
end.

```

При запуске полученной программы можно увидеть, как консольное окно появляется и тут же исчезает. Если внимательно присмотреться, можно увидеть, что в этом окне появляется надпись «Hello, world!», но совершенно очевидно, что не каждый пользователь успеет её разглядеть.

Почему так происходит, догадаться нетрудно: программа запускается, выводит сообщение и тут же завершает свою работу, а при завершении работы программы её окно должно исчезнуть с экрана. Получается, чтобы увидеть результат работы программы, надо заставить её выполняться чуть дольше. Например, до тех пор, пока пользователь не нажмёт какую-нибудь клавишу. Допишем программу следующим образом:

```

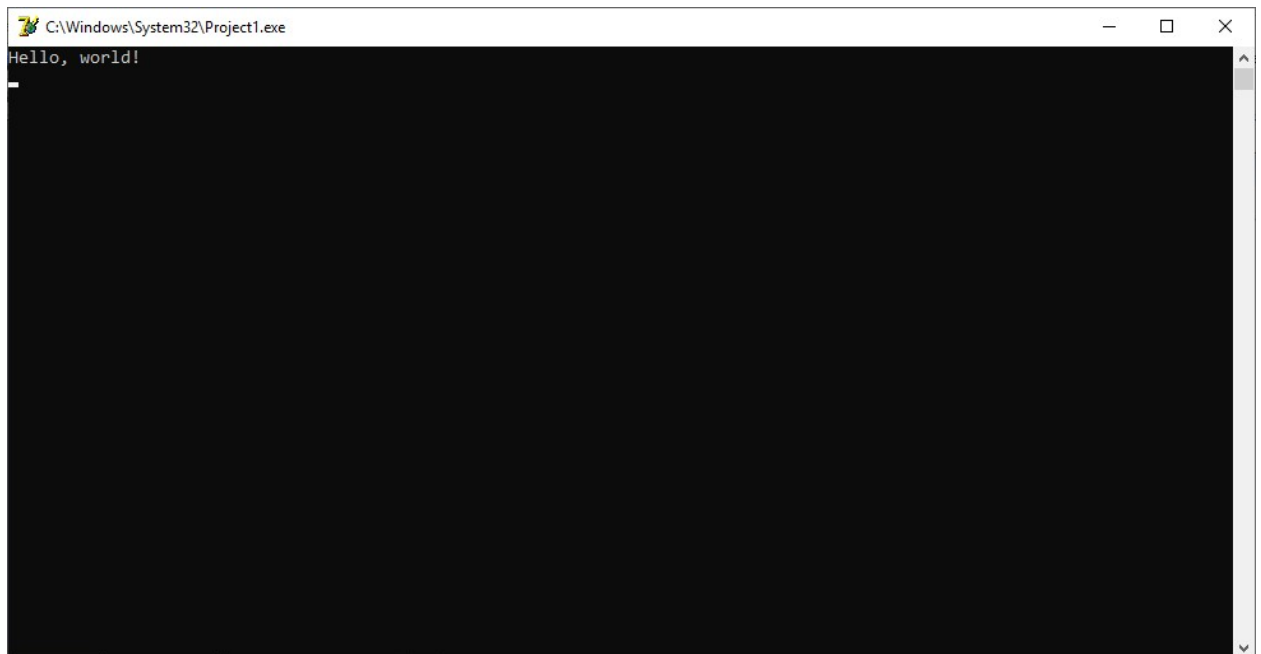
program Project1;

{$APPTYPE CONSOLE}

begin
    WriteLn('Hello, world!');
    ReadLn;
end.

```

Теперь программа запускается, отображает приветственное сообщение и ждёт. Чтобы завершить её, достаточно нажать клавишу Enter. Только после этого окно исчезнет и выполнение программы будет завершено. Программа работает, как и ожидается:



Ну что же, теперь давайте разбираться, что мы только что написали.

Самый простой способ вывести какую-либо информацию в Delphi — использовать встроенные процедуры `Write` и `WriteLn`. Обе эти процедуры принимают произвольное количество параметров и делают одно и то же — отображают значения параметров в том

же порядке, в котором они были переданы. Отличие заключается в том, что WriteLn после этого ещё перемещает курсор в начало следующей строки. В случае же процедуры Write курсор останется в позиции, следующей сразу за последним выведенным значением.

Кроме вывода информации обычно также требуется её ввод. Для этих целей в Delphi имеются встроенные процедуры Read и ReadLn. Они также принимают произвольное количество параметров, но теперь эти параметры не задают выводимую информацию, а принимают вводимую. Отличия между ними примерно те же, что и между Write и WriteLn.

Для взаимодействия с пользователем крайне Read используется крайне редко, т.к. не всегда её поведение будет ожидаемым как для пользователя, так и для начинающего программиста. Для полного понимания логики работы Read и её отличий от ReadLn необходимо поработать с файлами (именно там возможности Read раскрываются в полной мере), а также ознакомиться с механизмом буферизации ввода, используемым в компьютерах, но это более сложные темы, поэтому пока достаточно принять за правило использовать при взаимодействии с пользователем только ReadLn.

В нашем примере ReadLn была вызвана без параметров, поэтому просто дождалась подтверждения ввода. Давайте теперь попробуем всё-таки ввести какие-нибудь данные и что-нибудь с ними сделать. Например, решим такую задачу:

Написать программу, которая вычисляет сумму двух введенных пользователем целых чисел.

Проанализируем задачу. Исходные данные — два числа, введенные пользователем. Это будут переменные, которые мы назовём, скажем, X1 и X2. Результат — сумма. Для неё тоже отведём отдельную переменную, которую назовём Sum. Поскольку каких-то особых ограничений на значения переменных не накладывается, будет использовать тип Integer, как самый производительный из целочисленных типов.

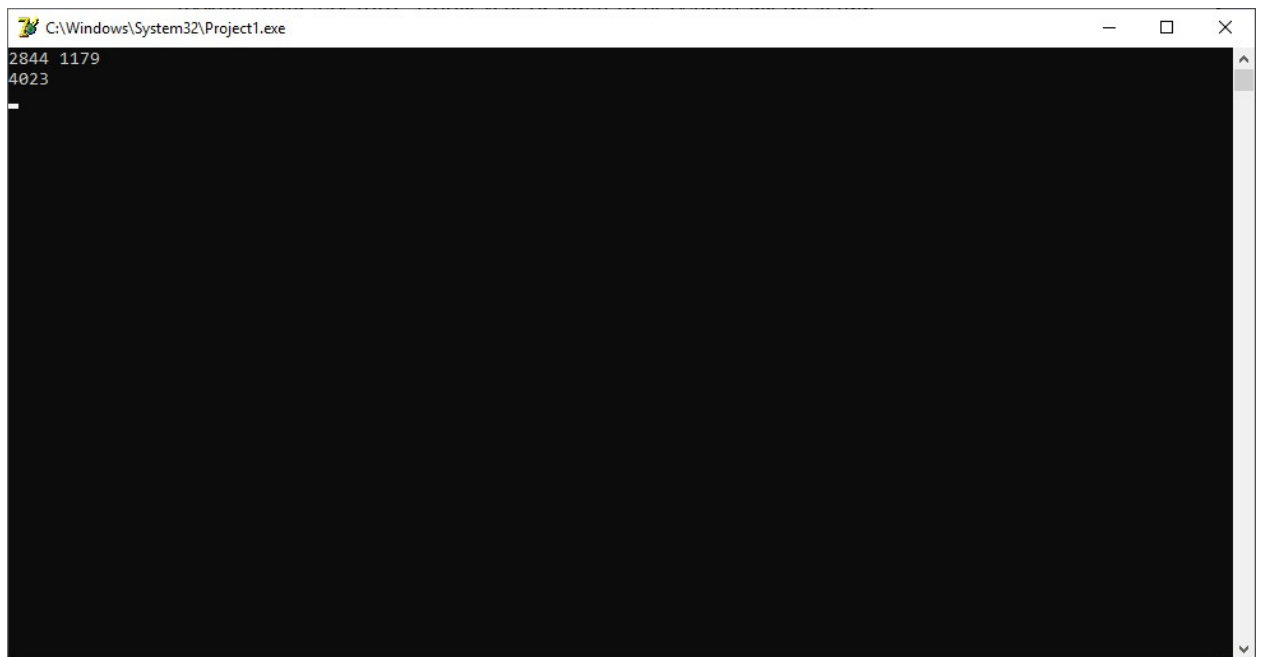
```
program Project1;

{$APPTYPE CONSOLE}

var
  X1, X2: Integer;
  Sum: Integer;

begin
  ReadLn(X1, X2);
  Sum := X1 + X2;
  WriteLn(Sum);
  ReadLn;
end.
```

Проанализируем полученную программу. В первой строке процедуре ReadLn переданы переменные X1 и X2. Ввод их значений будет ожидать в том же порядке (но для вычисления суммы порядок вообще-то не очень-то и важен). Во второй строке в переменную Sum помещается значение суммы X1 + X2. Процедура WriteLn, вызванная в третьей строке, отобразит значение переменной Sum. Четвёртая строка в этой программе нужна лишь для того, чтобы успеть увидеть результат вычислений.



В тех случаях, когда `ReadLn` вызывается с несколькими переменными в качестве параметров, можно вводить запрашиваемые значения, разделяя их как пробелами, так и нажатиями клавиши `Enter`.

На самом деле можно обойтись и без переменной `Sum`:

```
program Project1;  
  
{$APPTYPE CONSOLE}  
  
var  
    X1, X2: Integer;  
  
begin  
    ReadLn(X1, X2);  
    WriteLn(X1 + X2);  
    ReadLn;  
end.
```

Эта программа работает аналогично предыдущей, ошибок в ней нет, но злоупотреблять такими вещами не стоит: кажущаяся экономия в больших проектах может обернуться трудностями в сопровождении. В конце концов сегодня Вам нужно просто вывести данные, а завтра — ещё и пустить в дальнейшие вычисления, так что рано или поздно придётся объявить-таки эту переменную. А даже если и не придётся, документировать код, в котором вычисления и ввод/вывод выполняются отдельно, намного проще. Что же до экономии, то компиляторы уже давно достаточно умны, чтобы самостоятельно разбираться, какие переменные действительно нужны, а какие можно смело выбросить из программы, так что Ваши 4 байта, вполне вероятно, будут сэкономлены и без дополнительных усилий с Вашей стороны.

Нетрудно убедиться, что программа работает корректно (в пределах возможностей типа `Integer`). Однако кое-что в ней очевидно нехорошо. Что именно?

Следует помнить, что пользователь программы не видит её исходного кода. В нашей программе получается, что при её запуске пользователь остаётся один на один с пустым консольным окном. Программа ждёт действий пользователя, пользователь ждёт, чтобы

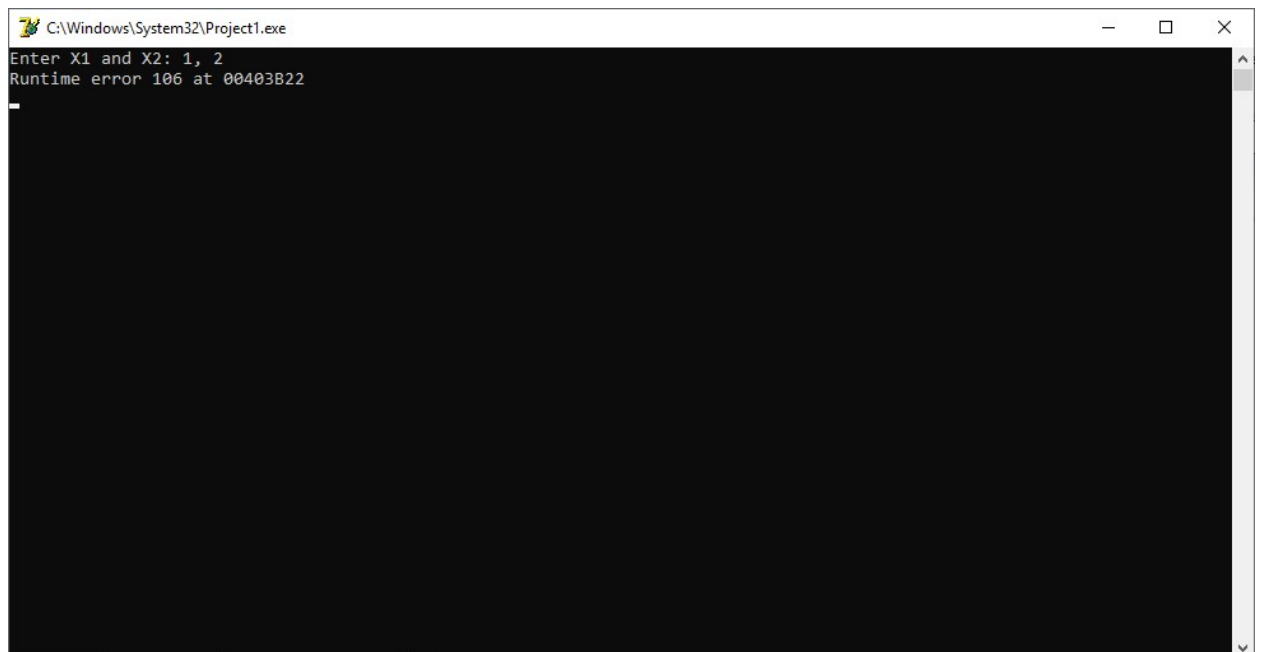
программа проявила свою готовность к работе. Предположим, по прошествии 5 минут пользователь догадался, что программа уже готова к работе и ждёт ввода каких-то данных. Каких? А что, если это не программа сложения двух чисел, а что-то посерьёзнее? А что, если эта программа посерьёзнее управляет работой атомной электростанции и от ввода пользователя зависит, как изменяются параметры её работы?

Повернём нашу программу лицом к людям:

```
program Project1;  
  
{$APPTYPE CONSOLE}  
  
var  
    X1, X2: Integer;  
    Sum: Integer;  
  
begin  
    Write('Enter X1 and X2: ');  
    ReadLn(X1, X2);  
    Sum := X1 + X2;  
    WriteLn(Sum);  
    ReadLn;  
end.
```

Хорошо ли программа написана сейчас?

Поставим себя снова на место пользователя. Программа запускается и просит ввести два значения — X1 и X2. Хорошо, предположим, что ввести их надо именно в таком порядке. Но чем разделить два значения? Вы же помните, что пользователь не видит исходного кода и тем более не знает, как работает ReadLn? А чем люди обычно разделяют несколько значений?..

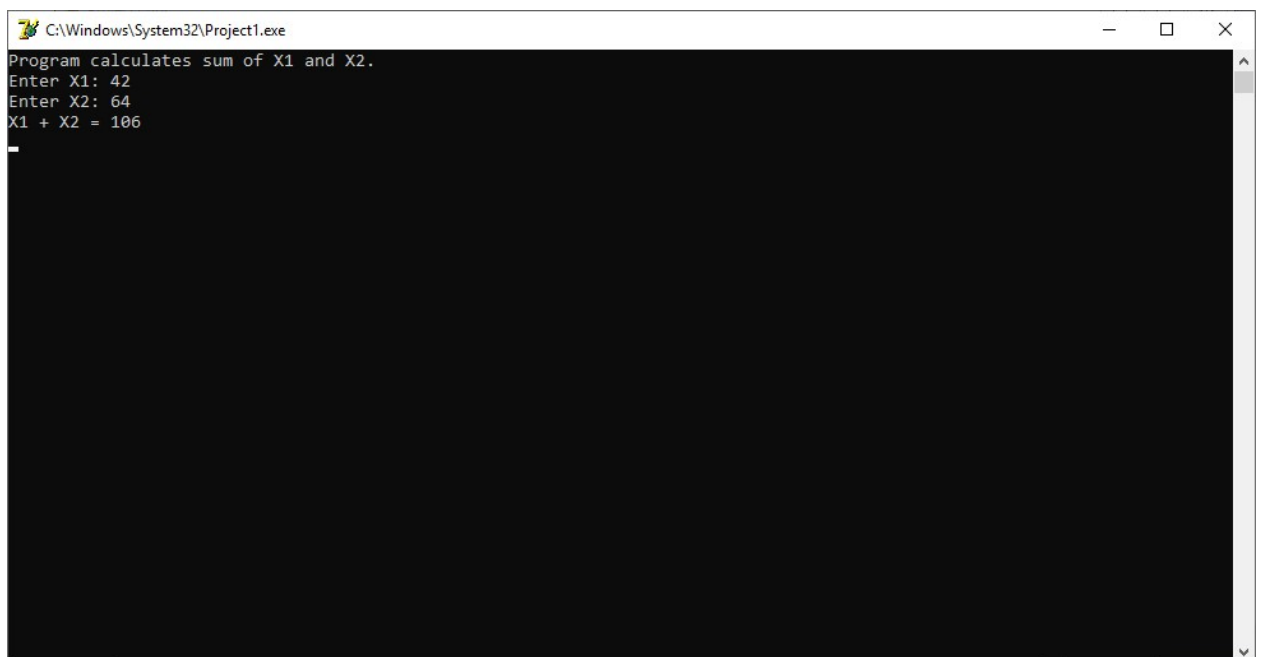


В следующую секунду окно исчезает. Скорее всего, пользователь даже не успел увидеть, что там появилось какое-то сообщение о какой-то ошибке, так что самое информативное сообщение, которое Вы получите, — «Программа не работает». Пользователь искренне уверен, что всё сделал правильно, поэтому, если не проявить чудеса догадливости, можно

долго пытаться понять, в чём дело. А ещё программами, которые не понимают пользователя, обычно не любят пользоваться.

Попробуем ещё раз:

```
program Project1;  
  
{$APPTYPE CONSOLE}  
  
var  
    X1, X2: Integer;  
    Sum: Integer;  
  
begin  
    WriteLn('Program calculates sum of X1 and X2.');    Write('Enter X1: ');  
    ReadLn(X1);  
    Write('Enter X2: ');  
    ReadLn(X2);  
  
    Sum := X1 + X2;  
  
    WriteLn('X1 + X2 = ', Sum);  
    ReadLn;  
end.
```



Стало действительно лучше, только вот вместо 3–4 строк в ней теперь все 8, да ещё и две пустые пришлось добавить, чтобы отделить ввод и вывод от собственно вычислений. Но для пользователя ничего не должно быть жалко: он и деньги платит, и популярность программе создаёт. Пользователю должно быть удобно. Даже ценой неудобства программиста.

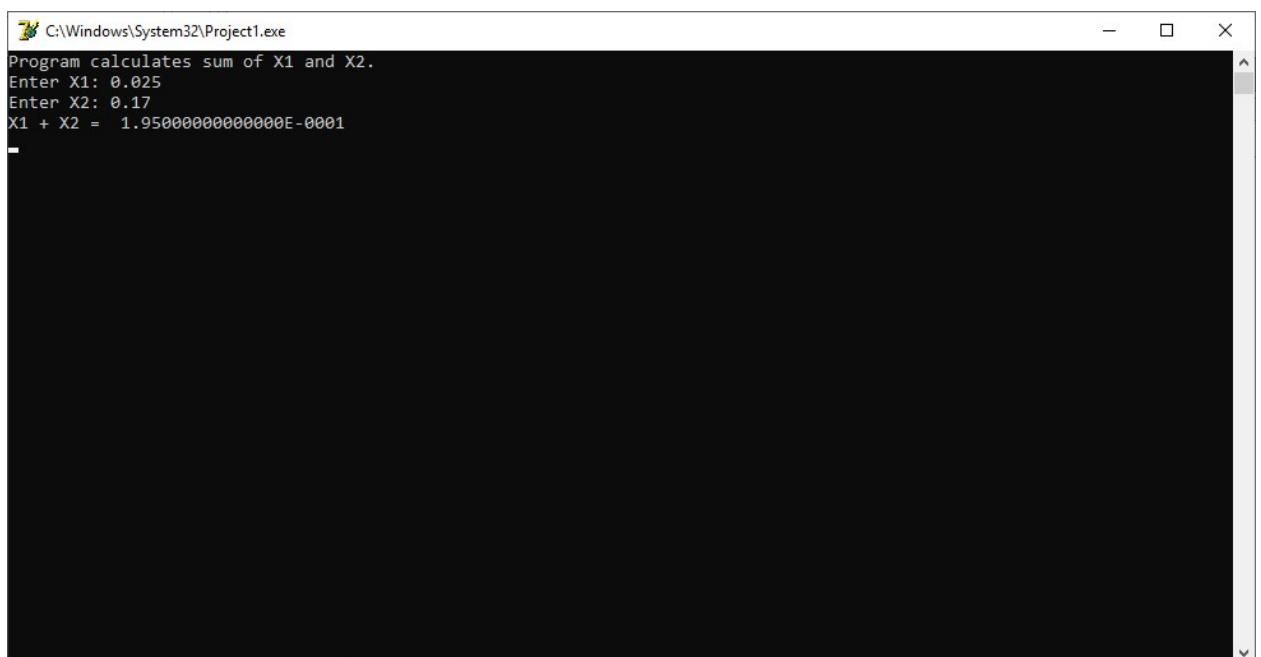
Но целые числа складывать всё же скучно. Давайте-ка немного усложним себе задачу:

Написать программу, которая вычисляет сумму двух введённых пользователем чисел.

Заметили разницу в постановке задачи? Всего-то пропало слово «целых». Но теперь мы не имеем права полагаться на то, что числа, вводимые пользователем, будут только целыми.

```
program Project1;  
  
{$APPTYPE CONSOLE}  
  
var  
    X1, X2: Real;  
    Sum: Real;  
  
begin  
    WriteLn('Program calculates sum of X1 and X2. ');  
    Write('Enter X1: ');  
    ReadLn(X1);  
    Write('Enter X2: ');  
    ReadLn(X2);  
  
    Sum := X1 + X2;  
  
    WriteLn('X1 + X2 = ', Sum);  
    ReadLn;  
end.
```

Легко, не правда ли? Всего-то заменить Integer на Real или любой другой вещественный тип. Ну что же, давайте посмотрим:



Знаете, что скажет на это среднестатистический пользователь? Что программа сломалась. Представьте себе, что такое число Вам отобразил банкомат, когда Вы пытались посмотреть остаток средств на Вашей банковской карте.

Чуть более продвинутый пользователь догадается, что перед ним какая-то странная форма записи числа и заметит, что цифры вроде бы совпадают с теми, которые должны быть. Но после этого всё равно попросит: «А можно мне как-то попроще эту информацию выводить? Я же по-человечески исходные данные вводил!»

Удобство пользователя надо уважать, помните? К счастью, здесь это будет не так уж сложно: нам на помощь придёт *форматированный вывод*.

Встроенные процедуры, работа которых связана с текстовым представлением числовых величин, поддерживают особый синтаксис передачи параметров. Предположим, пользователь хочет получить результат с точностью в 4 знака после запятой (обратите внимание на вывод результата):

```
program Project1;

{$APPTYPE CONSOLE}

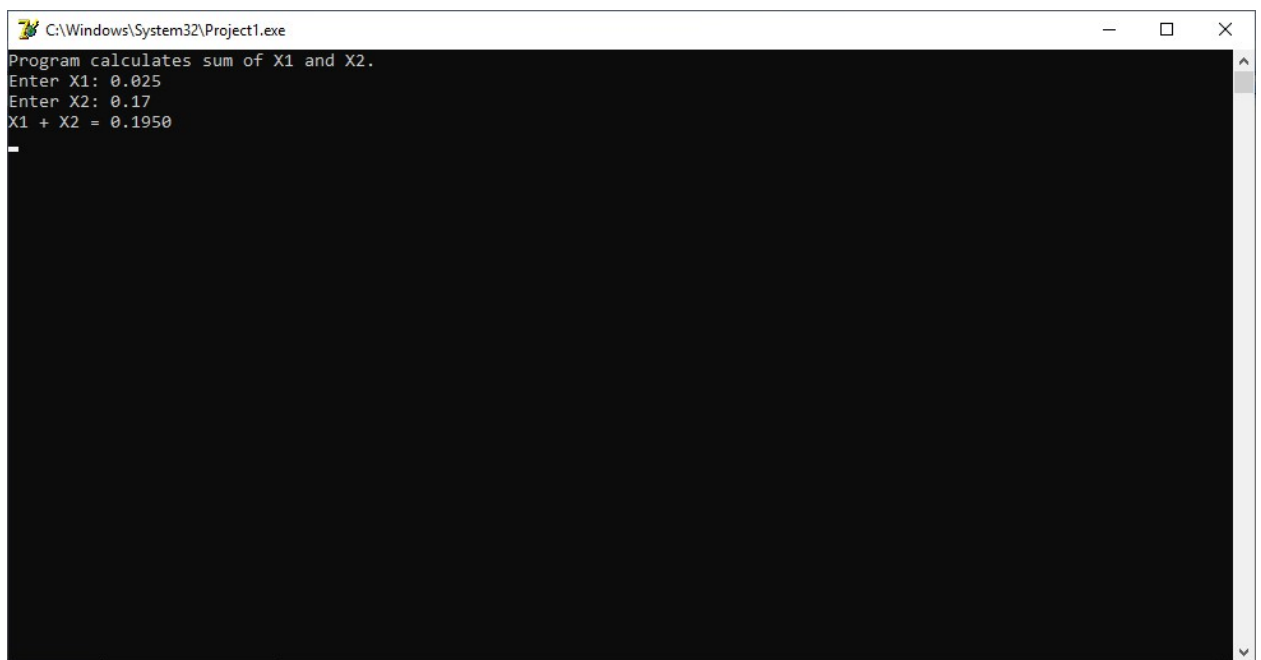
var
  X1, X2: Real;
  Sum: Real;

begin
  WriteLn('Program calculates sum of X1 and X2. ');
  Write('Enter X1: ');
  ReadLn(X1);
  Write('Enter X2: ');
  ReadLn(X2);

  Sum := X1 + X2;

  WriteLn('X1 + X2 = ', Sum:0:4);
  ReadLn;
end.
```

Результат работы программы для тех же данных теперь будет таким:



В общем виде запись такого параметра выглядит так:

```
OutExpr [: MinWidth [: DecPlaces ] ]
```

где `OutExpr` — это выражение, значение которого нужно вывести, `MinWidth` — минимальное количество позиций (знакомест), которое должна занять запись всего значения целиком, а `DecPlaces` — количество десятичных цифр после запятой. `MinWidth` и `DecPlaces` должны быть целочисленными выражениями. Если для записи значения требуется меньше позиций, чем задано `MinWidth`, спереди оно будет дополнено пробелами. `MinWidth` задаёт только минимальное количество знакомест: если

это значение слишком мало, чтобы корректно отобразить заданное значение, будет использовано большее количество места.

Например, в нашем примере `MinWidth` задано равным 0. Очевидно, ни одно число не будет корректно отображено в 0 знакомест, поэтому задание такого числа фактически означает «использовать столько места, сколько потребуется».

Оба значения указываются только для выражений вещественных типов. Для остальных типов применяется только указание минимального количества позиций. Например, для целочисленного значения можно записать так:

```
Write(X1:6, X2:6);
```

В этом случае каждое из чисел будет занимать на экране не менее 6 знакомест. Например, числа 8 и 16 будут отображены так:

```
_ _ _ _ _ 8 _ _ _ _ 16
```

Процедуры `Write` и `WriteLn` могут также выводить значения символьного и логического типов. Для символьного типа отображается соответствующий символ, для логического — текст «TRUE» или «FALSE», в зависимости от значения соответствующего параметра. Форматированный вывод в этом случае также применим.

Дополнительные вопросы

1. Напишите программу, которая для заданного пользователем числа N выводит сумму первых N натуральных чисел.
2. Напишите программу, которая для заданных пользователем координат двух точек выводит расстояние между этими точками с точностью в 4 знака после запятой.
3. Доработайте программу из п. 2 так, чтобы пользователь мог сам задавать количество знаков после запятой.