**55 Принцип модульности**

**Дополнительно 10 класс**

Рассмотрим первую задачу.

В 1742 году член Российской академии наук, математик Кристиан Гольдбах отправил письмо своему другу и коллеге, швейцарскому математику Леонарду Эйлеру, в котором привел гипотезу о том, что любое натуральное число не меньше 6 можно представить в виде суммы трех простых чисел. В ответ на это Эйлер заметил, что эту гипотезу можно свести к тому, что любое четное число не меньше 4 можно представить в виде суммы двух простых чисел. Полученная гипотеза была названа гипотезой Гольдбаха. Она остается и не подтвержденной, и не опровергнутой по сей день.

Напишем программу, которая для заданного четного натурального числа **n ≥ 4**, проверяет гипотезу Гольдбаха. Если гипотеза была подтверждена, нужно вывести простые числа, суммой которых является заданное **n**.

Для того, чтобы проверить эту гипотезу, мы будем перебирать простые числа, меньшие **n**, пока не найдем такое простое число **m**, при котором разность **n** и **m** также является простым числом. Если же, перебрав все простые числа, которые меньше **n**, мы не найдем **m**, удовлетворяющее заданному условию, гипотеза будет опровергнута.

Мы знаем, что простым называется натуральное число больше единицы, которое без остатка делится только на единицу и само на себя. И мы уже решали задачу по определению того, является ли целое число простым. Для этого нужно проверить, не делится ли число на два, а также на все нечетные числа до квадратного корня из себя самого.

Для решения задачи нам понадобятся две функции, первая из которых, будет проверять, является ли заданное число простым, а вторая будет искать первое простое число, следующее после заданного. Обе эти функции мы опишем в отдельном модуле, в котором будут храниться функции обработки простых чисел.

Начнем написание этого модуля. Назовем его **SimpleNum**.

1. Для начала опишем функцию, определяющую, является ли целое число простым. Назовем функцию **isSimple**. Это будет логическая функция, так как она вернет одно из двух значений типа **bool**. Названия логических функций обычно начинаются со слова **is**. У функции будет один аргумент – целое положительное число **n**.
2. В теле функции сначала проверим делимость **n** на 2. Для этого запишем ветвление с условием, что **n % 2 == 0** и **n != 2**, так как 2 – простое число. Если это условие выполняется, значит **n** не является простым числом, а далее будет следовать инструкция возврата, завершающая работу функции и возвращающая значение **False**.
3. Для расчета квадратного корня из **n** нам понадобится соответствующая функция из модуля **math**. Загрузим ее в описываемый модуль за пределами функции. Далее напишем цикл для проверки делимости **n** на нечетные числа от 3 до квадратного корня из него самого. Это будет цикл с параметром **i**, изменяющимся в диапазоне от 3 до целой части из квадратного корня из **n**, увеличенной на два, с шагом два. В теле цикла мы запишем ветвление с условием, что **n** **%** **i** **== 0**, при этом **n ! =i**. Если это условие выполняется, то мы завершим работу функции, вернув значение **False**, так как число **n** не является простым.
4. После цикла, если функция еще не завершила свою работу – число **n** является простым, и мы завершим работу функции, вернув значение **True**. На этом описание первой функции будет завершено.

Опишем функцию поиска минимального простого числа, следующего после **k**. Назовем ее **nextSimple**.

1. У нее будет один параметр – целое число **k**. В теле функции сначала увеличим значение **k** на единицу, так как нас интересуют числа после **k**.
2. Дальше проверим, не больше ли **k** 2. Если это так, то следующим будет минимальное простое число 2 и мы завершим работу функции, вернув это значение.
3. Если **k** больше 2, то мы продолжим поиск простого числа. Так как четное число, которое больше 2, не может быть простым, то мы будем искать его среди нечетных чисел. Поэтому проверим четность **k**, записав ветвление с условием, что **k % 2 == 0**; если это условие выполняется, перейдем к следующему нечетному числу, увеличив **k** на 1.
4. Далее запишем цикл для перебора нечетных чисел, пока одно из них не окажется простым. Это будет цикл **while**, который будет работать до тех пор, пока **k** не окажется простым числом. В качестве условия цикла с отрицанием укажем значение функции **isSimple (k)**. Тело цикла будет содержать всего одну команду – переход к следующему нечетному числу, то есть увеличение **k** на 2. Цикл завершит свою работу, когда число **k** будет простым. Поэтому после цикла завершим работу функции, вернув значение **k**. Мы завершили описание модуля. Сохраним его.

Теперь напишем модуль для решения задачи. Назовем его **Goldbach** и сохраним в одной папке с модулем **SimpleNum**.

1. Загрузим в наш модуль обе функции из модуля, описанного ранее.
2. С помощью инструкции **print** выведем на экран сообщение о том, что это программа, проверяющая гипотезу Гольдбаха для заданного **n**.
3. С помощью следующей инструкции **print** выведем на экран запрос на ввод **n** без перехода на следующую строку. Далее запишем инструкцию для считывания **n**. По условию задачи **n** – целое, поэтому при считывании преобразуем его в целочисленный тип **int**.
4. Теперь нам нужно найти такое простое **m < n**, при котором разность **n – m** также будет простым числом. Объявим переменную **m**, которой присвоим наименьшее простое число – 2. Дальше запишем цикл для поиска **m**, удовлетворяющего заданному условию. Он будет работать до тех пор, пока **m < n** и **n – m** не является простым числом. То есть вторая часть условия цикла – это с отрицанием значение функции **isSimple (n - m)**.
5. Тело цикла будет содержать единственную команду – присваивания переменной **m** следующего простого числа, то есть значения функции **nextSimple (m)**.
6. После завершения работы цикла, чтобы проверить, верна ли гипотеза, запишем ветвление с условием **m < n**. Если это условие выполняется, значит было найдено простое число **m**, удовлетворяющее условию задачи, и мы с помощью инструкции **print** выведем на экран сообщение о том, что для заданного числа **n** гипотеза подтверждена и что **n** равно сумме **m** и числа **n** минус **m**. Если же условие не выполняется, мы выведем на экран сообщение о том, что гипотеза опровергнута. На этом описание модуля завершено.

74 3 и 71

1024 3 и 1021

При решении задачи мы независимо друг от друга разработали модуль обработки простых чисел, а также основную программу, что соответствует важному принципу программирования – **модульности**. Он состоит в том, что сложные программы разрабатываются и тестируются по частям. При этом все модули независимы друг от друга и изменение одного из модулей не повлечет за собой изменение другого. Функции, которые мы описываем в модулях, могут быть применены при решении разных задач, поэтому написанные модули стоит сохранять и при необходимости расширять, чтобы не решать снова уже решенную задачу, если она встретится снова. Этот принцип называется **повторным использованием кода**. Эти принципы увеличивают количество доступных нам инструментов, сокращая при этом время, необходимое для написания программ.

Рассмотрим еще одну задачу.

В математике часто требуется разложить некоторое натуральное число на простые множители. Написать программу, представляющую заданное натуральное число **n**, которое будет не меньше 2, в виде произведения простых множителей.



Эту задачу можно решить следующим образом. Найти первое простое число, на которое без остатка делится заданное **n**. После этого делить заданное **n** на найденное число до тех пор, пока остаток от деления не станет равным 0, при этом увеличивая некоторую переменную-счетчик. Если на каком-то этапе то, что осталось от **n**, перестанет делиться нацело на найденное простое число, то мы выведем на экран первый множитель числа, то есть найденное простое число в степени, равной значению переменной-счетчика. После чего мы обнулим значение счетчика и найдем следующее простое число, на которое **n** все еще делится нацело, и повторим те же действия. Так будет продолжаться до тех пор, пока значение **n** не станет равным 1.

Начнем написание программы для решения задачи.

1. Наверняка многие догадались, что для этого нам будет полезна функция поиска следующего простого числа **nextSimple** из модуля **SimpleNum**, описанного ранее, поэтому загрузим ее в описываемый модуль.
2. С помощью инструкции **print** выведем на экран сообщение о том, что это программа для разложения числа **n** на простые множители.
3. С помощью следующей инструкции **print** выведем на экран запрос на ввод **n** без перехода на следующую строку. Дальше напишем инструкцию для считывания **n**. По условию задачи **n** – целое число, поэтому при считывании преобразуем его значение в целочисленный тип **int**.
4. Объявим переменную **m**, в которой будем хранить простые числа. Присвоим ей наименьшее простое число – 2.
5. С помощью инструкции **print** выведем на экран сообщение, состоящее из значения переменной **n** и знака равенства, а также объявим логическую переменную **p**, в которой будем хранить истинность высказывания о том, что ни одного простого множителя еще не найдено. Так как пока это правда, присвоим **p** значение **True**.
6. Далее напишем цикл для поиска множителей **n**. Это будет цикл **while**, который будет работать до тех пор, пока **n > 1**.
7. В цикле запишем ветвление с условием, что **n % m == 0**. Если это условие выполняется, объявим переменную-счетчик  **s** = 0, в ней будем рассчитывать степень найденного простого числа **m** в составе **n**.
8. Запишем для этого цикл. Он будет работать до тех пор, пока **n % m != 0**. В нем будем увеличивать значение счетчика **s** на 1, а **n** нацело делить на **m**.  По завершении работы цикла в переменной **s** будет содержаться степень **m**.
9. Запишем ветвление, которое проверяет, является ли найденное простое число **m** первым делителем **n**. Его условием будет значение логической переменной **p**. Если значение переменной **p** – «истина», то **m** – это первый найденный множитель **n**. Присвоим **p** значение **False**, после чего выведем на экран без перехода на следующую строку сообщение **m** **^** **s**, в противном случае **m** – не первый найденный множитель, и мы выведем на экран сообщение \* **m** **^** **s**. После внешнего ветвления присвоим переменной **m** значение следующего простого числа, найденного с помощью функции **nextSimple (m)**.

Мы завершили описание модуля. Запустим его на выполнение.

1024 2 ^ 10

7 7 ^ 1

1000 2 ^ 3 \* 5 ^ 3

**Мы узнали:**

·     **Модульностью** называется принцип, состоящий в том, что сложные программы составляются из независимых частей, которые разрабатываются и тестируются по отдельности.

·     **Принцип повторного использования кода** состоит в том, что стоит сохранять модули, в которых описаны функции для решения задач, для того чтобы не решать их в дальнейшем, если они встретятся еще раз.

·     Эти принципы увеличивают количество доступных инструментов и позволяют ускорить разработку новых программ.