Некоторые задачи отборочного этапа Олимпиады школьников "Надежда энергетики" по предмету "информатика" в 2014/2015 учебном году

Задание 1.

1. Все школьники знают, что такое простое число и различные алгоритмы проверки на простоту. Гипотеза о бесконечном числе простых чисел-близнецов утверждает: "Существует бесконечно много таких простых p, что и p+2 - тоже простое". Мы не просим Вас подтвердить или опровергнуть гипотезу. Вам предлагается разработать алгоритм для нахождения *простых чисел-близнецов*.

1)Тут надо учесть, что при проверке на простоту надо ограничить поиск делителей от 3 до sqrt(n). Алгоритм проверки на простоту можно выбрать следующий:

```
алг Простое (арг цел N)
нач
  цел i

  если N = 1 то
    вернуть ложь
  всё
  если N = 2 или  N = 3 или  N = 5 или  N = 7 то
    вернуть истина
  всё
  если N mod 2 = 0 то
    вернуть ложь
  всё
  если N mod 3 = 0 то
    вернуть ложь
  всё

  для i от 1 до целая_часть(sqrt(N)) div 6 + 1
  нц
    если N mod (6 * i - 1) = 0 то
      вернуть ложь
    всё
    если N mod (6 * i + 1) = 0 то
      вернуть ложь
    всё
  кц
  вернуть истина

кон
.
```

2) если внимательно прочитать задание, то алгоритм для нахождения экв-но поиску бесконечного числа простых чисел-близнецов. Тут нет конкретного исходного числа p. Поскольку алгоритм – это конечная последовательность шагов (в отличие от алгоритмической процедуры, где допускается зацикливание), то необходимо предусмотреть какую-то верхнюю границу. Например, задать число M: n <M. Тогда задание целесообразно использовать с выделением в алгоритме функции проверки числа на простоту и цикла.

Задание 2.

При малых значениях *x* математическая функция $f(x) = e^x$ может быть рассчитана как сумма ряда $f(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \cdots$. Пожалуйста, реализуйте алгоритм для расчёта функции *f* и шага *i*, на котором закончено вычисление с точностью *ε*. Условие останова: $\left|f(x_i) - f(x_{i-1})\right| \le \varepsilon$.

Разность между $f(x_{i-1})$ и $f(x_i)$ – это очередное слагаемое. Значит, вычисления можно прекращать, когда очередное слагаемое станет меньше заданной точности. Кроме того, чтобы сократить вычисления, нужно определить закономерность в слагаемых. В данном ряде каждое слагаемое описывается формулой $\frac{x^i}{i!}$. Первое слагаемое имеет номер 0: $\frac{x^0}{0!} = \frac{1}{1} = 1$, второе – номер 1: $\frac{x^1}{1!} = \frac{x}{1} = x$, и т.д. Для того чтобы определить так называемое *рекуррентное соотношение*, позволяющее вычислить *i*-ое слагаемое, зная (*i* – 1)-ое слагаемое, необходимо поделить формулу *i*-ого слагаемого на формулу (*i* – 1)-ого слагаемого: $\frac{x^i}{i!} : \frac{x^{i-1}}{(i-1)!} = \frac{x^i \cdot (i-1)!}{x^{i-1} \cdot i!} = \frac{x}{i}$. x^i и x^{i-1} сокращаются совершенно очевидным образом. С факториалами может быть чуть сложнее, но идея такая же: $i! = (i-1)! \cdot i$ (также как $x^i = x^{i-1} \cdot x$). Таким образом, если мы вычислили (*i* – 1)-ое слагаемое, мы можем умножить его на $\frac{x}{i}$ и получить *i*-ое слагаемое, и нам не надо каждый раз производить возведение в степень и вычислять факториал.

```
алг Экспонента(арг вещ x, вещ eps, рез вещ y, цел i)    // y – результат, i – номер слагаемого
нач
  вещ  s                              // s – очередное слагаемое

  s = 1                               // Первое слагаемое вычисляем вручную
  y = s                               // Записываем его в общую сумму
  i = 1                               // Далее будем вычислять второе слагаемое (с номером 1)
  пока abs(s) > eps                   // Если x отрицательно, слагаемое тоже может быть отрицательным
  нц
    s = s * x / i                     // Вычисляем i-ое слагаемое, используя рекуррентное соотношение
    y = y + s                         // Прибавляем его в общую сумму
    i = i + 1                         // Увеличиваем номер слагаемого
  кц
кон
```

Задание 3.

При малых значениях x математическая функция $f(x) = 1 - \sqrt[4]{1-x}$ может быть рассчитана как сумма ряда $f(x) = \frac{x}{4} + \frac{3 \cdot x^2}{4 \cdot 8} + \frac{3 \cdot 7 \cdot x^3}{4 \cdot 8 \cdot 12} + \ldots$. Пожалуйста, реализуйте алгоритм для расчёта функции *f* и шага *i*, на котором закончено вычисление с точностью *ε*. Условие останова: $\left|f(x_i) - f(x_{i-1})\right| \le \varepsilon$.

Разность между $f(x_{i-1})$ и $f(x_i)$ – это очередное слагаемое. Значит, вычисления можно прекращать, когда очередное слагаемое станет меньше заданной точности. Кроме того, чтобы сократить вычисления нужно определить закономерность в слагаемых и вычислить так называемое *рекуррентное соотношение*, позволяющее подсчитать *i*-ое слагаемое, зная (*i* – 1)-ое слагаемое. В данном ряде мы видим, что степень *x* возрастает на 1 в каждом последующем слагаемом. Кроме того, в каждом следующем слагаемом появляется один дополнительный сомножитель в числителе и один дополнительный сомножитель в знаменателе. Можно увидеть, что последний сомножитель в знаменателе описывается формулой $4 \cdot i$, а последний сомножитель в числителе описывает формулой $(4 \cdot i - 5)$, где i – номер слагаемого. Таким образом, если мы вычислили $(i - 1)$-ое слагаемое, мы можем умножить его на $\frac{(4 \cdot i - 5) \cdot x}{4 \cdot i}$ и получить i-ое слагаемое, и нам не надо каждый раз производить возведение в степень и вычислять произведение многих сомножителей.

```
алг СуммаРяда(арг вещ x, вещ eps, рез вещ y, цел i)     // y – результат, i – номер слагаемого
нач
  вещ s                                           // s – очередное слагаемое

  s = x / 4                                       // Первое слагаемое вычисляем вручную
  y = s                                           // Записываем его в общую сумму
  i = 2                                           // Далее будем вычислять второе слагаемое
  пока abs(s) > eps                               //  Если  x  отрицательно,  слагаемое  тоже  может  быть
отрицательным
  нц
    s = s * x * (4 * i – 5) / 4 / i               //  Вычисляем  i-ое  слагаемое,  используя  рекуррентное
соотношение
    y = y + s                                     // Прибавляем его в общую сумму
    i = i + 1                                     // Увеличиваем номер слагаемого
  кц
кон
```

Задание 4.

Разработать алгоритм вычисления $\sqrt{2}$, используя непрерывную дробь $\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \ldots}}}$, с точностью 0.0001.

Прежде всего, заметим, что самое первое слагаемое не укладывается в общую схему. Поэтому будем вычислять дробь $2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \ldots}}}$, которая окажется на 1 больше, чем $\sqrt{2}$. Эту единицу надо будет потом вычесть из полученного результата.

Далее, в данном случае, в отличие от суммы ряда, мы не можем вычислить, какой именно вклад даёт каждый следующий уровень дроби. Поэтому необходимо вычислять i-ое и $(i + 1)$-ое приближения, и прекращать вычисления, когда модуль разности между ними станет меньше точности.

Кроме того, данную дробь можно вычислить только с конца. Прежде чем поделить 1 на что-то, надо вычислить, на что именно её надо поделить. Поэтому мы должны зафиксировать количество уровней дроби и только после этого вычислять это приближение.

```
/* Первая функция просто вызывает вспомогательную функцию, которая будет вычислять приближения, начиная с
одного уровня дроби. Пользователь не должен знать о том, что нужно передавать некий параметр, да ещё и
вычитать 1. Корень из 2, сам по себе, не зависит ни от каких вспомогательных параметров. */
алг sqrt2()
нач
  вернуть sqrt2_2(1) – 1
кон

/* Вторая функция вычисляет приближение, которое получается при использовании n уровней дроби, и
приближение, которое получается при использовании n + 1 уровней дроби. Если разность между этими двумя
приближениями меньше требуемой точности, вычисления прекращаются. Иначе мы рекурсивно вызываем эту же
функцию, увеличив параметр (число уровней дроби) на 1. */
```

```
алг sqrt2_2(цел n)
нач
  вещ f1, f2
  f1 = sqrt2_3(n, 1)
  f2 = sqrt2_3(n + 1, 1)
  если abs(f1 – f2) < 0.0001
    вернуть f2
  иначе
    вернуть sqrt2_2(n + 1)
  всё
кон

/* Третья функция собственно вычисляет требуемую дробь. Первый параметр – нужное количество уровней, второй параметр – достигнутое количество уровней. */
алг sqrt2_3(цел n, цел k)
нач
  если n = k
    вернуть 2 + 1.0 / 2
  иначе
    вернуть 2 + 1.0 / sqrt2_3(n, k + 1)
  всё
кон
```

В предложенном решении можно заметить существенный недостаток. Функция *sqrt2_2* при первом вызове вычисляет 1-ое и 2-ое приближения, при втором вызове – 2-ое (которое уже было вычислено) и 3-е приближения и т.д. Лучше было бы передавать уже вычисленное приближение на следующий уровень рекурсии.

```
/* Теперь мы вызываем вспомогательную функцию, которой передаём два параметра, - первое приближение, которое мы вычислили вручную, и уровень дроби, который теперь равен 2, т.к. первое приближение уже вычислено. */
алг sqrt2()
нач
  вернуть sqrt2_2(2.5, 2) – 1
кон

// Почти то же самое, что и в предыдущем варианте, только f1 не вычисляется, а передаётся.
алг sqrt2_2(вещ f1, цел n)
нач
  вещ f2
  f2 = sqrt2_3(n, 1)
  если abs(f1 – f2) < 0.0001
    вернуть f2
  иначе
    вернуть sqrt2_2(f2, n + 1)
  всё
кон

// Третья функция осталась без изменений.
алг sqrt2_3(цел n, цел k)
нач
  если n = k
    вернуть 2 + 1.0 / 2
  иначе
    вернуть 2 + 1.0 / sqrt2_3(n, k + 1)
  всё
кон
```

Ещё одно возможное изменение – в ряде случаев (и в этом, в том числе) рекурсию можно заменить итерацией. Это всегда полезно с точки зрения уменьшения количества используемых ресурсов.

```
// Первая функция вобрала в себя вторую, которой теперь нет.
алг sqrt2()
нач
  вещ f1, f2
  цел n

  f1 = 0                        // Любое число, которое существенно отличается от первого приближения
  f2 = 2.5                      // Первое приближение
  n = 2                         // Будем вычислять дробь с двумя уровнями
  пока abs(f1 – f2) > 0.0001
  нц
```

```
    f1 = f2                     // Сохраняем старое приближение
    f2 = sqrt2_3(n)             // Вычисляем новое приближение
    n = n + 1
  кц
  вернуть f2 - 1
кон

// В третьей функции рекурсия заменена на итерацию.
алг sqrt2_3(цел n)
нач
  вещ f
  цел i

  f = 2 + 1.0 / 2
  для i от 1 до n – 1
  нц
    f = 2 + 1.0 / f
  кц
  вернуть f
кон
```

Если очень внимательно посмотреть на непрерывную дробь, которую мы вычисляем, то можно заметить, что все уровни одинаковы. Обозначим за a_n приближение, получающееся при вычислении n уровней дроби. Тогда $a_{n+1} = 2 + 1 / a_n$. Поэтому $\sqrt{2}$ можно вычислить более простым способом.

```
алг sqrt2()
нач
  вещ f1, f2

  f1 = 0
  f2 = 2.5
  пока abs(f1 – f2) > 0.0001
  нц
    f1 = f2
    f2 = 2 + 1.0 / f2
  кц
  вернуть f2 - 1
кон
```

Задания 5, 6.

Школьник Алексей хочет упростить работу с обыкновенными дробями. Для этого он решил считать операции с дробями на компьютере. Помогите Алексею – разработайте алгоритм деления обыкновенных дробей *a/b* и *c/d*. Если при делении результатом является неправильная дробь, то ответ представить в виде смешанного числа.

Школьник Фёдор хочет упростить работу с обыкновенными дробями. Для этого он решил считать операции с дробями на компьютере. Помогите Фёдору – разработайте алгоритм умножения обыкновенных дробей *a/b* и *c/d*. Если при умножении результатом является неправильная дробь, то ответ представить в виде смешанного числа.

После деления/умножения необходимо найти наибольший общий делитель числителя и знаменателя, сократить дробь, выделить из неё целую часть, и если она не равна 0, пересчитать числитель.

```
алг Дроби(арг цел a, цел b, цел c, цел d, рез цел intPart, цел numerator, цел denominator)
нач
  цел nod1, nod2

  a = a * d                                        // Для деления
  b = b * c

  a = a * c                                        // Для умножения
  b = b * d

  если a = 0 то                                    //  Если  числитель  равен  0,  то  мы  не  сможем
вычислить
    intPart = 0                                    // наибольший общий делитель
    numerator = 0
  иначе
    если b = 0 то                                  //  Если  знаменатель  равен  0,  то  присваиваем
значение -1
      intPart = -1                                 // всем переменным, как признак ошибки
      numerator = -1
      denominator = -1
    иначе
      nod1 = abs(a)                                // Находим наибольший общий делитель
      nod2 = abs(b)
      пока nod1 <> nod2
      нц
        если nod1 > nod2 то
          nod1 = nod1 – nod2
        иначе
          nod2 = nod2 – nod1
        всё
      кц
      a = a / nod1                                 // Сокращаем числитель и знаменатель
      b = b / nod1
      если a < 0 и b < 0 или a >= 0 и b < 0 то     // Нормализуем знак «минус»
        a = -a
        b = -b
      всё
      intPart = a div b                            // Находим целую часть
      если intPart <> 0 то                         // Если она не равна 0,
        numerator = abs(a) – abs(intPart) * b      // пересчитываем числитель
      иначе
        numerator = a
      всё
      denominator = b
    всё
  всё
кон
```