В помощь учителю. Основы олимпиадного программирования. Часть I.

Пирогов В.Ю.

Содержание

Введение

Многолетний опыт организации олимпиад по программированию навел меня на мысль, что неудачи на олимпиадах и школьников и студентов часто связаны не с отсутствием у участников способностей к решению олимпиадных задач, а с неудовлетворительной начальной подготовкой. Многие, как правило, не знают, как организовать входной и выходной поток данных в программе, тогда как условия предлагаемых задач ориентированы именно на такой интерфейс. Многие задачи на олимпиадах для школьников имеют вполне типовую основу, но если школьник никогда не

решал ничего подобного, то эти задачи станут для него непреодолимым препятствием и, увы, закроют дорогу в мир программирования.

Данное пособие рассчитано как раз на начальный этап подготовки будущих участников олимпиад по программированию. На мой взгляд, для некоторых учащихся этого уже вполне достаточно, чтобы пуститься в самостоятельное плавание и в дальнейшем самому повышать свой уровень программного мастерства. Я надеюсь, что данное пособие будет полезно учителям информатики, позволит им посмотреть более осмысленно на подготовку их учеников в области программирования и в частности олимпиадного программирования.

Когда-то олимпиады по программированию, были единственными олимпиадами в области информационных технологий. В настоящее время даже олимпиад по программированию существует несколько видов: классическая олимпиада, олимпиады по спортивному программированию, олимпиады по проектированию программного обеспечения, олимпиады - марафоны, олимпиады по программированию на конкретных языках программирования и т.д. Наше пособие ориентировано на классический тип олимпиады, на который, в частности, ориентируются и устроители школьных олимпиад по программированию.

Олимпиадное программирование

Правила олимпиад

Олимпиада – это соревнование, сравнение результатов разных участников, с целью выявления лучших показателей. Для того чтобы результаты были объективными, нужны общие правила, которым подчиняются и участники, и устроители олимпиады. Правила – это всегда некоторые ограничения. Рассмотрим некоторые типовые правила, которые имеются практически на всех современных олимпиадах по программированию, о которых и участники и тот, кто готовит участников, должны знать.

Язык программирования, системы программирования

Одним из самых важных вопросов, который возникают каждый раз, когда задумываешься об организации олимпиады по программированию – это какие языки и системы программирования будут допускаться к использованию. Существует огромное количество языков программирования. Современные продвинутые молодые люди часто самостоятельно изучают языки. Нужно ли ограничивать участников в выборе языка? Опыт показывает, что это делать необходимо, если мы хотим, чтобы результаты олимпиады был объективны, по крайней мере, стремились к объективности. Есть два важных параметра, которые влияют на такой выбор.

Во-первых, время выполнения программы, зависит не только от выбранного (разработанного) участником алгоритма, но и от выбранного компилятора (интерпретатора), которые в свою очередь зависят от языка программирования. Если мы не будем ставить ограничения на язык программирования, то не сможем сформулировать требования по ограничению времени выполнения программ. Программа, написанная, скажем на языке Python, в которой участник нашел новое оригинальное решение, может выполняться много медленнее, программ, написанных на языке C, просто по причине того, что программы на языке Python интерпретируются. С другой стороны определить для множества языков и трансляторов (которыми могут воспользоваться участники) интервал времени выполнения программы - абсолютно не выполнимая задача для любых организаторов олимпиады любого уровня.

Во-вторых, в состав систем программирования могут входить различные библиотеки, которые могут значительно облегчить задачу написания программы. Особенно это касается различных функций анализа и обработки текстовой информации, функций сортировок, вычисления различных не стандартных математических функций и т.д. В результате участник, в распоряжении которого будут иметься такие средства программирования, получит преимущество по отношению к другим участникам. Учесть все это для множества возможных языков программирования также практически не возможно.

> ***Замечание***
>
> *Второй аргумент, пожалуй, сильнее первого, так как в отсутствии возможности для участников проверки решений на эталонном компьютере, жесткого ограничения по времени выполнения программы вводить не следует.*

Все это приводит к одному очень важному выводу: количество языков программирования (и компиляторов) должно быть жестко ограничено. Олимпиады высокого уровня, как правило, ограничивают язык, на котором могут писать участники следующим набором: C, C++ иногда Java, в качестве исключения также Pascal. На олимпиадах низкого уровня (школьные, районные, городские) количество разрешенных языков может оказаться гораздо больше. Продиктовано это тем, что не все участники имеют специализированную подготовку по указанным выше языкам и чтобы не сводить к минимуму количество участвующих, им разрешается писать программы на разных языках и использовать разные системы программирования.

При подготовке же будущих участников олимпиады нельзя не учитывать тот случай, когда они получат высокие результаты и им все равно придется использовать языки, предлагаемые устроителями олимпиады более высокого уровня, на которые им придется ехать. Таким образом, подготовка должна ориентироваться на один из четырех языков программирования: C, C++, Java, Pascal. Лучше, если это будут языки C или C++.

К участию в олимпиадах, даже самого низкого уровня надо готовить!

Время, отводимое на решение задач

При организации олимпиад по программированию делается оценка времени, которое необходимо участникам для решения. Не обязательно расчет ведется из возможности решить все задачи. Участник может выбирать задачи из некоторого списка и, оценивая время решения, свои силы и количество очков, которые он хочет заработать, строить некоторую траекторию достижения своих целей (последовательность решения задач и время, затраченное на каждую задачу). Умения планировать свои действия на олимпиаде приходит постепенно с опытом участия в подобных мероприятиях. Олимпиады с укороченным временем (много задач и мало времени) обычно называют спортивным программированием.

Оформление задач

Обычно обязательным требованием к оформлению программ является подпись автора программы: ФИО, город, место учебы. Организаторам олимпиады важно иметь соответствие между текстом программы и участником соревнований. В действительности эта информация является дополнительной, поскольку каждому зарегистрированному участнику (или команде) выделяется свой вычислительный ресурс: компьютер (ы), виртуальная машина (ы) или место на сетевом диске.

Что касается текста самой программы, то здесь к участникам требования не предъявляются, если только сама олимпиада не предполагает какие либо специфические особенности написания про-

грамм. Впрочем, красиво (читабельно) оформлять программу и писать информативные комментарии в ней, является одним из важных качеств любого программиста, в том числе и решателя сложных олимпиадных задач. Если дело дойдет до разбора алгоритма программы (например, при возникновении спора), то хорошо прокомментированная программа будет иметь преимущество перед программой без комментариев.

Время выполнения программ

Время выполнения программы важный показатель правильности выполнения задания. В олимпиадах высокого уровня обычно указывается верхний предел (например, 2 с.), превышение которого автоматически означает, что данное решение не прошло данный тест. При жестком ограничении на время выполнения программы должен быть тестовый компьютер, на котором и проверяются все задачи. Соответственно тестовые программы, которые имеются у организаторов олимпиады, прогоняются на этом компьютере и являются источником определения верхней границы по времени. В случае использования специальных автоматизированных олимпиадных систем участники олимпиады могут отослать задание на эту системы, которая сразу покажет результат: количество пройденных тестов. Таким образом, каждый участник может в реальном времени отлаживать свою программу. При наличии автоматических систем может вводиться еще одно ограничение – это используемый программой объем память. При оперировании большими объемами информации, а также при использовании рекурсивных алгоритмов это может быть существенным показателем.

В случае, если возможности тестирования программы во время ее решения нет, то обычно вводят более мягкое ограничение по времени. Его также привязывают ко времени выполнения имеющихся у организаторов олимпиады решений. Однако в качестве верхнего предела может браться удвоенное или утроенное время выполнения тестовой программы. Это необходимо, чтобы члены жюри имели конкретные ориентиры во время проверки заданий (нельзя же ждать произвольно долго, когда программа закончит свою работу).

Проверка заданий

Есть два подхода к проверке олимпиадных заданий. Первый подход, я бы назвал ручной проверкой. Он предполагает не только проверку правильности работы программы, но и возможный анализ текста программы. В пользу этого подхода говорят многочисленные случаи на олимпиадах, когда не хватает буквально нескольких минут, чтобы дорешать задачу. Анализ текста программы в таком случае позволил бы оценить работу участника олимпиады. На первый взгляд это справедливо, но привносит в соревнование элемент субъективизма. Ведь кто-то должен оценивать текст программы, которая, в общем-то, не работает. На мой взгляд, ручная проверка вполне допустима на олимпиадах низкого уровня (в рамках школы, района, одного Вуза). Ценность такого подхода можно усилить, введя возможность для авторов программ защищать свои проекты, не все, но те, для которых как раз возникает проблема анализа текста программы.

Второй подход полностью формален. Однако формальность подхода позволяет минимизировать субъективность оценки, а также упрощает проверку заданий в случае большого числа участников. Каждая задача оценивается по результатам прохождения тестов. На основе этих результатов затем форсируется оценка. Например, если максимальная оценка m, а пройдено n тестов из N, результирующая оценка может высчитываться по простой формуле m*(n/N). Реже, тесты могут иметь вес, и тогда оценка будет вычисляться несколько сложнее: $(\sum n_i * p_i)/N$, где n_i=0,1 (результаты тестов) и $\sum p_i$ = 1 – условие для весов. Для самих тестов при этом задается граничное время выполнения программы и, возможно, максимальная память, которую использует програм-

ма. Т.е. тест считается пройденным, когда на выходе программы получаем правильные данные, программа не превысила лимит времени выполнения и не затратила памяти больше, чем оговаривается в условии.

Подробнее вопрос о тестировании олимпиадных задач будет изложен во второй части нашего пособия.

Требование к интерфейсу программ

Перейдем, пожалуй, к самому важному требованию, которое обычно предъявляется на всех олимпиадах по программированию. В основу решения любой олимпиадной задачи лежит принцип входной и выходной информации: программа получает некоторый набор данных (тест) на входе и в результате работы должна выдать решение, т.е. так же набор данных. Приведу пример типичного условия олимпиадной задачи:

Задача 1. Количество равных из трех (100 баллов)
Ограничения: время – 250ms, память – 64M
Входные данные.
Даны три целых числа (числа, не превосходящие 10^9), записанных в отдельных строках. Определите, сколько среди них совпадающих.
Выходные данные.
Программа должна вывести одно из чисел: 3 (если все совпадают), 2 (если два совпадают) или 0 (если все числа различны).
Пример

Ввод	**Вывод**
1 2 2	2

Не разбирая решение приведенной задачи, остановимся только на ее условии. Отметим, что раздел **Ограничения** имеет смысл применять, только при условии наличия эталонного компьютера, доступ к которому должны иметь все участники олимпиады во время ее проведения. Другими словами должна быть некоторая автоматизированная проверяющая система, доступная всем участникам. Если такой системы нет, то, как было сказано ранее, можно использовать мягкое ограничение по времени и не вводить ограничение по используемой памяти.

В условии задачи есть также разделы **Входные данные** и **Выходные данные**. В абсолютном большинстве олимпиад по программированию речь в них идет о структуре входного текстового файла и структуре выходного текстового файла. Эта унификация необходима для упрощения и автоматизации (по возможности) проверки выполнения заданий. Ниже в таблице приводится пример того, что должен содержать входной и выходной файлы. Наиболее удобный и приемлемый подход заключается в том, что программа должна получать данные во входном потоке и выводить результат в выходной поток. Такая возможность есть во всех основных операционных системах и предусмотрена в языках программирования. Для этого программа:

1. Должна иметь законченный бинарный вид исполняемого модуля для данной операционной системы. Если используется интерпретирующий язык, то программа должна представлять собой модуль, который может быть запущен с помощью консольного интерпретатора.
2. Должна быть консольной (не графической!!!).

3. Должна использовать для ввода и вывода информации стандартные функции ввода и вывода на консоль. Другими словами программа не должна быть диалоговой, а спроектирована для использования в пакетном режиме.

> ***Замечание***
>
> *Конечно, с точки зрения обучения программированию вообще, очень важна такая тема как «Разработка интерфейса пользователя». Но она не имеет отношения к вопросам, которые рассматриваются в данном пособии. Участники олимпиад должны знать и использовать принципы пакетной обработки.*

Рассмотрим принципы работы программ рассчитанных на обработку стандартных потоков ввода-вывода. Пусть исполняемый модуль называется program.exe. В листинге 1 представлены возможные варианты запуска программы с использованием перенаправление ввода-вывода.

Листинг 1. Консольная программа program.exe, работает со стандартными устройствами ввода-вывода

```
program.exe  - означает ввод входных данных с консоли, вывод выходных данных на консоль.

program.exe < input.txt  – ввод входных данных из текстового файла input.txt,  вывод выходных данных на консоль.

program.exe > output.txt  - ввод входных данных с консоли, вывод выходных данных в текстовый файл output.txt.

program.exe > output.txt < input.txt  -  ввод входных данных из текстового файла input.txt, вывод выходных данных в текстовый файл output.txt.
```

> ***Замечание***
>
> *Для работы в консоли нужны определенные навыки, которых так не хватает современному пользователю. Для упрощения освоения командной строки мы рекомендуем использовать программу far для Windows и программу MC для всех Unix-подобных систем.*

Давайте теперь перейдем к двум конкретным примерам того, как организуется программа для обработки входных-выходных потоков данных. Представленные ниже примеры предполагают обработку входных файлов с произвольным количеством строк. Я надеюсь, что основываясь на этих примерах, читатель без труда напишет программы, для обработки ситуаций, когда строго задается количество значимых строк во входном файле, или когда разные строки во входном файле носят разные смысловые нагрузки. Оба примера продублированы на трех языках программирования: C, C++, Pascal, с использованием только стандартных возможностей.

В листинге 2 представлена программа, читающая произвольное количество строк во входящем потоке и выводящая эти строки в выходной поток. В качестве языка программирования выбран язык C. После компилирования, исполняемый модуль может быть использован по схеме, представленной в листинге 1.

Листинг 2. Обработка произвольного количества строковой информации (язык C)

```
#include <stdio.h>
int main() {
      char s[1000];
      while(gets(s)){
/*здесь выполняем какие либо действия с полученной строкой*/
```

```
/*выводим строку для контроля*/
      puts(s);
     }
     return 0;
}
```

Комментарий к листингу 2.

Для ввода данных со стандартного устройства мы используем стандартную функцию языка С gets. Функция читает строку со стандартного устройства (stdin), до появления в потоке символа перевода строки или конца файла. Считанная строка копируется в буфер, указателем на который является параметром функции. Функция считается «не безопасной», так как в ней отсутствует контроль длины вводимой строки. Обычно, рекомендуется использовать ее аналог функцию fgets, но мы не будем останавливаться на этом, поскольку на рассматриваемые вопросы это никак не влияет. Функция возвращает указатель на считанную строку; в случае достижения конца файла или ошибки, функция возвращает нулевое значение (указатель NULL). Это и обыгрывается в представленном алгоритме: while(gets(s)) – «пока не равно нулю».

Для вывода строки на стандартное (stdout) устройство используется также стандартная функция puts. Программа будет работать до тех пор, пока не кончаться строки, не будет нажато сочетание клавиш ctrl+c – прерывание работы программы. Если ввод осуществляется с консоли, то для окончания ввода можно нажать ctrl+z, что вызывает посылку на стандартное устройство ввода символа конца файла.

Программа более ничего не содержит. Вы можете наполнить ее собственным содержанием: ввести любую обработку вводимой строки, а также добавить дополнительные условия выхода из цикла. Обратим внимание, что если функция gets не помещает в результирующую строку символа перевода строки и символа конца файла, то функция puts автоматически добавляет к концу выводимой строки символ перевода строки ('\n').

Обратимся теперь к программе из листинга 3. Программа в точности повторяет логику работы программы из листинга 2. Но для считывания и вывода строки используются другие средства (средства С++).

Листинг 3. Обработка произвольного количества строковой информации (язык С++)

```
#include <iostream>
using namespace std;
int main() {
      char s[1000];
      while(!cin.eof()){
      cin.getline(s,1000,'\n');
/*здесь выполняем какие либо действия с полученной строкой*/

/*выводим строку для контроля*/
      cout << s <<endl;
     };
     return 0;
}
```

Комментарий к листингу 3.

В листинге 3 также используются только стандартные средства, но языка С++. Основой организации ввода – вывода для стандартных устройств (по умолчанию консоль) здесь можно использо-

вать библиотечные объекты cin и cout. Для ввода строки со стандартного устройства (по умолчанию stdin) используется метод getline. Ввод может быть перенаправлен (см. Листинг 1). Обратим внимание, что в нем (методе getline), кроме указателя на строку (ср. с функцией gets из листинга 2) содержится максимальная длина вводимой строки и символ, дойдя до которого чтение должно остановиться: '\n' - символ перевода строки. Для проверки конца файла используется метод eof(): while(!cin.eof()) – пока не конец файла (потока).

Для вывода строки на стандартное устройство используется объект cout и перегруженный оператор «<». Строка endl соответствует просто символу '\n'. По умолчанию вывод осуществляется на консоль, но может быть перенаправлен (см. Листинг 1).

Обратимся теперь к листингу 4, в котором мы представляем аналог уже разобранных из листингов 2 и 3 программ, но написанный, теперь уже на стандартном языке Pascal.

Листинг 4. Обработка произвольного количества строковой информации (язык Pascal)

```
var
s:ansistring;
begin
     while(not eof) do
     begin
          readln(s);
{здесь выполняем, какие либо действия с полученной строкой}

{выводим строку для контроля}
          writeln(s);
    end;
end.
```

Комментарий к листингу 4.

В программе из листинга 4 для управления вводом – выводом строковой информации на стандартные устройства (с возможностью перенаправления, как это показано в листинге 1) используются стандартные функции readln и writeln. Для проверки достижения конца файла используется функция eof, в таком виде она проверяет стандартный поток ввода на его окончание. При этом readln читает строку, со стандартного устройства пока не встретит символ конца строки или конец файла. Функция writeln – пишет строку на стандартное устройство и добавляет в конец ее символ перевода строки. Функция eof возвращает значение true, если достигается конец файла.

> ***Замечание***
>
> *В программе, представленной в листинге 4, используется строковый тип ansistring, который есть и в последних версиях Free Pascal и в Delphi и, который допускает произвольную длину строки. В pascalABC вместо типа ansistring следует использовать обычный тип string, длина которого также не ограничена. Я очень надеюсь, что Turbo Pascal уже нигде не используется.*

Разобранные в листингах 2-4 примеры являются универсальными:

1. Вводимая строка может иметь сложную структуру, например, содержать подстроки, числа, флаги, вспомогательные символы. В этом случае в алгоритм вводится код для разбора вводимой строки – выделение из нее ее отдельных частей. Это не всегда удобно, поэтому обычно ввод данных сразу адаптируется под условие задачи (см. примеры из листингов 5-7).
2. Если по условию задачи на входе имеется заданное количество строк (см. например, условие задачи в разделе *Требование к интерфейсу программ*), при этом значение строки зависит от

ее номера, то всегда можно ввести переменную - счетчик. В зависимости от значения счетчика по-разному будут обрабатываться вводимые строки, а также будет осуществлен выход из цикла, если значение счетчика пересечет указанную границу или во вводимой информации будет обнаружена ошибка.

Как я уже отметил в том случае, если по условию задачи строка имеет сложную структуру, то можно провести разбор этой структуры (синтаксический разбор строки называется парсингом). С другой стороны, для разработки такого алгоритма может потребоваться некоторое количество времени, которого на олимпиаде почти всегда не хватает. В этом случае ввод данных сразу ориентируют на конкретную структуру на входе. Ниже (см. Листинги 5-7) мы рассматриваем обработку входных данных, представляющих произвольное количество строк, каждая из которых состоит из трех компонентов: два целых числа и одно вещественное число двойной точности (в языке С этот тип называется double).

И так структура входного файла имеет такой вид

```
123  345 678.8
45678 3145 2678.8
2 145 618.67
3 3145 22678.81
890 3225 6781.822
```

Замечу также, что между числами может стоять произвольное число пробелов.

Листинг 5. Обработка произвольного количества строк, содержащих числовые данные (язык С)

```
#include <stdio.h>
int main() {
      int a,b;
      double f;
      while(scanf("%d%d%lf",&a,&b,&f)!=-1){
/*здесь выполняем, какие либо действия с полученными числами*/

/*выводим числа для контроля*/
      printf("%d %d %lf\n",a,b,f);
    }
    return 0;
}
```

Комментарий к листингу 5.

В листинге 5 для ввода данных используется очень полезная стандартная функция scanf. Алгоритм ввода данных фактически задается форматной строкой "%d%d%lf". Функция scanf пропускает пробелы, пока ей не встретится число. Ввод числа заканчивается, когда функция встречает пробел (точнее любой символ, отличный от цифры). Для вещественных чисел (в форматной строке lf) функция учитывает наличие точки. Функция scanf возвращает -1, в случае ошибки, в том числе и появления символа конца файла: while(scanf("%d%d%lf",&a,&b,&f)!=-1). Для вывода чисел удобно воспользоваться универсальной функцией printf. Функция scanf и printf по умолчанию использует стандартные устройства ввода – вывода и допускают перенаправление (как это предполагается в листинге 1).

Листинг 6. Обработка произвольного количества строк, содержащих числовые данные (язык С++)

```
#include <iostream>
using namespace std;
int main() {
    int a,b;
    double f;
    while(!cin.eof()){
     cin >> a >> b >> f;
/*здесь выполняем, какие либо действия с полученными числами*/

/*выводим числа для контроля*/
     cout << a << ' ' << b<< ' '  << f <<endl;
    };
    return 0;
}
```

Комментарий к листингу 6.

В листинге 6 мы опять встречаемся со стандартными объектами cin (для ввода) и cout (для вывода). Метод eof объекта cin используется для определения конца файла. Как мы видим, для ввода сразу нескольких чисел мы использовали перегруженный оператор «>>». Для вывода данных (трех чисел в строке) на стандартное устройство используется перегруженный оператор «<<».

Листинг 7. Обработка произвольного количества строк, содержащих числовые данные (язык Pascal)

```
var
a,b:integer;
f:real;
begin
     while(not eof) do
     begin
          readln(a,b,f);
{здесь выполняем, какие либо действия с полученными числами}

{выводим числа для контроля}
          writeln(a,b,f);
     end;
end.
```

Комментарий к листингу 7.

Меньше всего (по сравнению с листингом 4) изменилась программа на Паскале. Универсальные функции readln и writeln позволяют также легко, как строки вводить и выводить числовые данные.

Примеры задач

В данном разделе мы рассмотрим несколько типичных задач олимпиадного характера уровня средней школы. Разобьем эти задачи на несколько разделов.

Числа

Задача 1. Вычитания

Условие.

Заданы два числа. До тех пор, пока оба они больше нуля, с ними производят одну и ту же операцию: из большего числа вычитают меньшее. Если числа равны, то из одного вычитают другое. Например, из пары (4,17) за одну операцию получается пара (4,13), а из пары (5,5) пара (0,5).

Вам задано некоторое количество пар (a_i, b_i). Сколько операций будет выполнено для каждой из них?

Входные данные.

Во входном файле n строк, каждая содержит пару целых положительных чисел a_i, b_i ($1 <= a_i, b_i <= 109$).

Выходные данные.

Искомое количество операций для каждой пары на отдельной строке.

Пример.

На входе:

100 1
2345678 2
4 17
7 987654321

На выходе:

100
1172839
8
141093479

Решение.

Представим решение задачи на двух языках (C и Pascal), см. Листинг 8. Программа может быть использована по схеме из листинга 1.

Листинг 8. Решение задачи 1 (вычитание) на языке C

```
#include <stdio.h>
int main() {
      int a,b,i;
      while(scanf("%d%d",&a,&b)!=-1){
/*обработка полученной пары*/
            i=0;
            while(a!=0&&b!=0){
                  if(a>b)a=a-b;
                  else b=b-a;
                  i++;
            }
/*выводим результат*/
      printf("%d\n",i);
      }
      return 0;
}
```

Комментарий к листингу 8.

В данном примере (листинг 8) следует обратить внимание на два вложенных цикла. Вообще, вложенные циклы очень часто встречаются в задачах олимпиадного типа. В данном случае внешний цикл считывает данные с внешнего устройства и для каждой пары чисел проводит операцию по-

следовательного вычитания (внутренний цикл), пока одно из чисел ни станет равным нулю. Переменная i играет роль счетчика.

В листинге 9 та же задача (задача 1) решена на языке Pascal.

Листинг 9. Решение задачи 1 (вычитание) на языке Pascal

```
var
      a,b,i:longint;
begin
      while(not eof) do
      begin
            readln(a,b);
{обработка полученной пары}
            i:=0;
            while((a<>0) and (b<>0)) do
            begin
                  if(a>b) then a:=a-b
                  else b:=b-a;
                  inc(i);
            end;
{выводим результат}
            writeln(i);
      end;
end.
```

Комментарий к листингу 9.

Листинг 9 полностью аналогичен листингу 8, так что никаких новых комментариев по поводу решение задачи на языке Pascal делать не нужно.

> ***Замечание***
>
> *Обратимся еще раз к задаче 1 (вычитание). Фактически речь в ней идет об алгоритме Эвклида нахождения наибольшего общего делителя (НОД) двух чисел. Только в постановке задачи речь идет не о нахождении самого общего делителя, а о нахождении количества шагов в таком алгоритме.*

Задания

1. Видоизменить программы из листингов 8 и 9 так, чтобы они кроме количества шагов также выдавали значение наибольшего общего делителя (НОД).
2. Перепишите программы из листингов 8 и 9, используя для внутреннего цикла операторы “do – while” и “for” (для программы на языке C) и “repeat – until” (для программы на языке Pascal).

Типичными задачами на олимпиадах по программированию являются задачи на решение равенств и неравенств на множестве целых чисел.

*Задача 2. Решение неравенства x*x+y*y<n*

Условие.

Дано натуральное n. Подсчитать количество решений неравенства x*x + y*y < n в натуральных (неотрицательных целых) числах, не используя действий с вещественными числами.

Входные данные.

Число n

Выходные данные.

Количество решений

Например.

На входе:

2

На выходе:

3

Решение задачи представим на языке С (см. Листинг 10).

Листинг 10. Решение задачи 2 на языке С

```
#include <stdio.h>
int main() {
      int s,n,i,l,i1;
      scanf("%d",&n);
      i=0; s=0;i1=0;
      while(i1<n){
            l=0;
/*получить количество решений при фиксированном i*/
            while(i1+l*l<n){
                  l++;
            }
            i++; i1=i*i; s=s+l;
      }
/*выводим результат*/
      printf("%d\n",s);
      return 0;
}
```

Комментарий к листингу 10.

В листинге программы мы опять видим вложенные циклы. Внешний цикл фиксирует левое слагаемое (i1), внутренний же цикл ищет все возможные решения неравенства при фиксированном левом слагаемом. Обратите внимание на переменную s, в ней накапливается суммарное значение количества всех решений неравенства. В переменной l хранится промежуточное значение – количество решений для конкретного значения левого слагаемого.

Задания

1. Перевести программу из листинга 10 на язык Pascal.
2. Видоизменить программу, так, чтобы она выдавала решения не неравенства, а равенства x*x + y*y = n.
3. Видоизменить программу так, чтобы она также выдавала и все решения неравенства, т.е. пары чисел x,y.

Обработка текстовой информации

Существует бесчисленное множество задач, связанных с обработкой текстовой информации. Это задачи поиска, сортировки, синтаксического разбора, кодирования, сжатия и т.д. Все эти тему могут стать источником формулирования олимпиадных задач.

> ***Замечание***
>
> *Современные языковые средства снабжены различными библиотеками, предназначенными для обработки текстовой информации. Эти библиотеки могут значительно облегчить задачу участников олимпиады, кто хорошо ими владеет, и поставить их в не равное положение по отношению к остальным. Здесь используются разные подходы. Можно в условии задачи запретить пользоваться подобными библиотеками, но тогда придется контролировать возможное их использование путем просмотра текста программ. Можно «изъять» подобные библио-*

теки, но технически это не всегда просто. Наконец, можно некоторые задачи, на обработку текстовой информации, заменить на подобные же задачи, где вместо текста используются числовые массивы.

Задача 3. Поиск подстроки в строке

Условие.

Дана строка s длины l, 0< l < 100000, и строка ss, длины 0<ll <= l.

Найти количество вхождений строки ss в строку.

Входные данные.

Две текстовых строки.

Выходные данные.

Число вхождений.

Пример.

На входе:

ffghjkllljgdfllj;;laall

ll

На выходе:

4

Решение задачи представим на языке С (см. Листинг 11).

Листинг 11. Решение задачи 3 (поиск подстроки в строке) на языке С

```
#include <stdio.h>
#include <string.h>
int main() {
      char s[100000];
      char ss[100000];
      int n=0,l,ll,i,j,k;
      gets(s); l=strlen(s);
      gets(ss); ll=strlen(ss);
      for(i=0; i<l-ll+1; i++){
/*найти вхождение первого символа второй строки в первой*/
            if(s[i]==ss[0]){
                  k=0;
/*проверка, не получили ли мы вхождение*/
                  for(j=i+1; j<i+ll; j++){
                        if(s[j]!=ss[j-i]){
                              k=1; //подняли флажок
                              break;
                        }
                  }
/*если k=0, то вхождение было*/
                  if(!k)n++;
            }
      }
/*выводим количество вхождений*/
      printf("%d\n",n);
      return 0;
}
```

Комментарий к листингу 11.

Обратим, прежде всего, внимание, что в классическом языке С строки представляют собой, по сути, обычные числовые массивы, так что решение, которое мы видим в программе, основывается на манипулирование массивами. Единственно, что нам потребовалось знать, это длины массивов и для этого мы воспользовались стандартной функцией strlen, которая никак не упрощает дальнейший алгоритм поиска (см. раздел Задания). Принцип поиска, который представлен в программе, прост: ищется первый символ второй строки в первой строке. Если символ находится, то далее осуществляется последовательная проверка всех символов второй строки на равенство с соответствующими символами первой (см. внутренний цикл). Далее поиск продолжается с символа, следующего за найденным символом (индекс i). Обратим также внимание на роль переменной k. Эта переменная принимает значение 1, если сравнение второй строки и участка первой строки закончилась неудачей.

Задания

1. Перепишите программу из листинга 11 так, чтобы не использовать библиотечную функцию strlen, основываясь на том факте, что строка в языке С заканчивается символом код, которого равен нулю.
2. Переведите программу из листинга 11 на язык Pascal. В программе не должны использоваться стандартные функции поиска подстроки в строке (кроме функции получения длины строки).
3. Объясните заголовки для внутреннего и внешнего цикла программы из листинга 11. Не запуская программы, объясните ее поведение (будет ли она выдавать правильный результат) в случае, когда вторая строка будет длиннее первой.
4. Перепишите программу из листинга 11, изменив требования поиска: если подстрока найдена, то следующий поиск начинается с символа, который не входит в данный результат поиска (участок первой строки, который совпал со второй строкой).

Рекурсивные задачи

Введение

Почти во всех олимпиадах есть задачи, которые решаются с использованием рекурсивных методов (по крайней мере, использование рекурсии сильно упрощает решение).

В программировании рекурсией называют вызов из процедуры самой себя. Различают простую и сложную рекурсии. В простой рекурсии вызов некоторой процедуры P1 осуществляется непосредственно из нее самой же. В сложной рекурсии из процедуры P1 вызывается другая процедура P2, а уже из нее вызывается процедура P1. Цепочка может быть более длинной: P1 ->P2 ->P3 ->…-> Pn->P1.

Рекурсивные алгоритмы в обычных алгоритмических языках сильно «съедают» стековую память, за счет того, что, по крайней мере, адрес возврата сохраняется в стеке. При большом количестве вызовов (без возврата) объем стека может быть исчерпан. По этой причине, если есть возможность, рекурсивные алгоритмы заменяют обычными алгоритмами. Кроме того, стараются предусмотреть в алгоритме различные механизмы, позволяющие сократить количество рекурсивных вызовов.

Приведем простой рекурсивный алгоритм вычисления факториала.

Листинг 12. Пример рекурсивной функции вычисления факториала на языке Pascal

```
function fact1(n : longint) : longint;
begin
```

```
      if n <= 1 then
            fact1:=1
      else
            fact1:=n*fact1(n - 1);
end;
```

Комментарий к листингу 12.

Рекурсивный алгоритм труднее понять, чем обычный (см. Листинг 13). В данном случае все не так уж сложно, но следует обратить внимание на то, как происходит возврат из рекурсии. Ведь бесконечная рекурсия не возможна, просто по причине переполнения стека. Кроме того в данной задаче мы и вычисляем факториал конечного числа. Заметим, что на каждом шаге рекурсии аргумент функции fact уменьшается, когда он достигнет 1, то происходит возврат из рекурсии. При этом функции возвращает значение, накопленное в fact. Заметим, что в качестве переменной-накопителя можно было использовать и глобальную переменную (что чаще всего и делают). Хочу также обратить внимание на еще один интересный момент. Функция возвращает значение, имеющее тип longint. Это и понятно, ведь факториал может достигать очень больших значений. Но в качестве аргумента функции мы могли бы взять и значение, имеющее тип integer. На первый взгляд мы могли бы сэкономить какое-то количество памяти, ведь аргумент функции стал короче. В действительности это не так. Дело в том, что в стек, скорее всего, в обоих случаях будет отправлено значение размерности longint.

Следует отметить несколько важных особенностей, которые нужны для понимания примеров рекурсивного программирования и написания собственных рекурсивных программ:

1. **Роль параметров рекурсивной функции.** Параметры рекурсивной функции (если они есть) вызывают функцию как бы в новом контексте. Очень часто параметром является некий номер, который вызывается с одновременным уменьшением на 1 (см. Листинг 12, а также листинг 15), что является также и «гарантией» выхода из рекурсии после n вызовов.
2. **Роль локальных переменных.** Поскольку локальные переменные также сохраняются в стеке, то при возврате из функции вся информация, которая была сохранена в них, не изменяется. Вместе с параметрами рекурсивной функции локальные переменные составляют контекст очередного рекурсивного шага.
3. **Роль глобальных переменных.** Почти всегда в рекурсивных алгоритмах требуется иметь нечто, что накапливается или вычисляется. Такого типа данные не всегда удобно хранить, например, в параметрах функции, так как во многих случаях они не должны изменяться и при возврате из функции. Кроме этого дополнительный параметр увеличивает количество стековой памяти, необходимой для вызова функции. Приходится хранить такие данные в глобальных переменных.
4. **Возврат из рекурсии.** При разработке рекурсивного алгоритма нужно четко представить себе, при каких условиях будет происходить возврат из рекурсии. Разработка таких условий часто требует довольно длительного времени.

Рассмотрим теперь функцию на языке Pascal, которая также вычисляет факториал, но не использует при этом рекурсию.

Листинг 13. Пример не рекурсивной функции вычисления факториала на языке Pascal

```
function fact2(n : longint) : longint;
      var
            i: longint;
```

```
          f: longint;
begin
      f:= 1;
      for i:= 2 to n do
            f:= f * i;
      fact2:= f
end;
```

Комментарий к листингу 13.

Обратим внимание, что не рекурсивный алгоритм, хотя и менее компактен, зато более понятен и не «съедает» стековую память. Сразу возникает вопрос: а почему во всех случаях ни использовать не рекурсивные алгоритмы? Дело в том, рекурсивные алгоритмы не только более компактны, но часто лежат на поверхности, как бы вытекает из самого условия задачи, тогда как для написания не рекурсивного алгоритмы надо иногда изрядно поломать голову.

Задача 4. Игра в числа (взята из олимпиады ШГПИ)

Условие.

Пусть дано число N и число M (M<N). Два игрока, назовем их A и B играют в следующую игру. Каждый из игроков по очереди называет число от 1 до M. Названные числа складываются. Выигрывает тот из игроков, чей очередной ход даст в результате сумму равную N. Например: N=10, M=5. A–4, B-2, A-4, т.о. A выиграл. Написать программу, которая на входе содержит числа N M (входной поток содержит в первой строке два числа через пробел), а на выходе (в выходном потоке) содержит все возможные варианты игры игроков A и B, в которых выигрывает A (A начинает играть). Не должны учитываться неправдоподобные варианты, когда игрок может выиграть за один ход, но не делает это. Например, при N=10, M=5 не должен учитываться вариант A-4, B-2, A-2, B-2.

Примеры.

При входных значениях 8 6

Имеем на выходе

1 1 6

1 2 5

1 3 4

1 4 3

1 5 2

1 6 1

При входных параметрах 6 2

Получим

1 1 1 1 2

1 1 1 2 1

2 2 2

Листинг 14. Пример рекурсии для задачи 4 (Игра в числа). Язык C.

```
#include <stdio.h>

int p1[100];
int p2[100];
int h1,h2,s;
```

```
int n,m;
/*печатает результат (когда A победил)*/
void write(){
      int i;
      for(i=0; i<h2; i++)printf("%d %d ",p1[i],p2[i]);
      printf("%d\n",p1[h1]);
};
/*рекурсивная функция поиска решений*/
void igra(){
      int i,j;
      for(i=1; i<=m; i++){
            s=s+i;
      //первый игрок закончил?
            if(s==n){
                  p1[h1]=i;   write();
                  s=s-i;  break;
            }
      //ход не годится?
            if(s>n){
                  s=s-i;  break;
            }
      //второй игрок может закончить следующим ходом?
      //если да, то дальше нет смысла играть
            if(n-s<=m){
                  s=s-i;
                  continue;
            }
      //ходит второй игрок
      //перебор возможных ходов
            for(j=1; j<=m; j++){
                  s=s+j;
                  p1[h1]=i; p2[h2]=j;
                  h1++; h2++;
                  igra(); //рекурсия
                  h1--; h2--;
                  s=s-j;
            }
            s=s-i;
      }
      return;
}
int main() {
      int i,s;
      scanf("%d%d",&n,&m);
      s=0;h1=0;h2=0;
      if(n>m) igra();
      return 0;
}
```

Комментарий к листингу 14.

Основой алгоритма получения вариантов игры является рекурсивная функция igra. Для того, чтобы понять как она работает, изучим в начале используемые в программе глобальные переменные. Глобальность переменных n и m в данном случае не обязательна, поскольку их значение можно было бы передавать через стек. Используя глобальные переменные, мы тем самым уменьшили задействованный объем памяти при каждом вызове функции igra. Глобальная переменная s используется для хранения текущего значения набираемой игроками суммы. Она меняется при каждом ходе игрока, а также при возврате из рекурсивной процедуры (чтобы вернуться на шаг назад). В глобальных массивах p1 и p2 программа хранит текущую цепочку ходов для каждого из игроков. При вызове функции write именно эти массивы используются для вывода последовательности ходов каждого игрока. Наконец, переменные h1 и h2 содержат в себе номер элемента массивов p1 и p2 соответственно, куда будет занесен очередной ход игрока. Разумеется, мы ограничиваем количество элементов в массиве, просто предполагая, что используемы для игры числа, не превосходят 100.

При разборе функции igra, следует обратить внимание на отсечение тривиальных ходов, т.е. ходов, когда с очевидностью один из игроков выигрывает сразу. Это несколько усложняет программу, зато отсекает лишние шаги в рекурсии и, соответственно, уменьшает время выполнения программы. Смысл самой рекурсии заключается в следующем: запуск функции igra – это фактически перебор ходов игрока A (внешний цикл). Ниже в функции имеется цикл (он вложенный), где перебираются возможные ходы игрока B, и при выборе хода опять запускается функция igra, т.е. опять выбираются возможные ответные ходы игрока A. Обратим также внимание, что при возвращении из функции igra в исходное состояние возвращаются переменные s,h1,h2.

Задания

1. Переведите программу из листинга 14 на язык Pascal.
2. Внесите изменения в программу, заменив переменные h1,h2 одной переменной h.
3. Попробуйте переписать программу так, чтобы она давала варианты выигрыша и для игрока A и для игрока B.

Комбинаторные задачи

Под комбинаторикой обычно понимают раздел математики, который изучает дискретные множества и различные сочетания элементов этих множеств. Значительная часть комбинаторики посвящена разработке алгоритмов генерации различных состояний этих множеств. Только с появлением компьютерной техники появилась возможность реализации подобных алгоритмов для достаточно больших множеств. Задачи с элементами комбинаторики часто встречаются на олимпиадах различных уровней.

Задача 5. Генерация перестановок (лексикографическая)

В данной задаче мы разберем один из алгоритмов генерации перестановок некоторого множества. Как мы знаем, количество таких перестановок равно n!. Вычислением самого числа n! мы уже занимались (см. Листинги 12 и 13). Получение всех генераций перестановок вещь куда более сложная. Мы немного отходим от того, что делали до сих пор: разбирали конкретные задачи олимпиад по программированию. Данная задача в некотором смысле задачей уже не является, поскольку алгоритмы генерации перестановок уже давно вошли во все учебники по программируемой комбинаторике и подготовка будущего участника олимпиад по программированию должна включать изучение подобных алгоритмов. Мы также приводим один из алгоритмов генерации

перестановок, реализованный на языке С (см. Листинг 15). Обращаем внимание, что здесь опять используется рекурсия.

Листинг 15. Генерация перестановок (язык С). Задача 5.

```
#include <stdio.h>
int p[20]; //массив
int m; //количество элементов в массиве
/*вывод массива p*/
void write(){
      int k1;
      for(k1=0; k1<m; k1++)printf("%d ",p[k1]);
      printf("\n");
}
/*произвести реверсию части массива*/
void reverse(int n){
      int k1,k2,p1;
      k1=0; k2=n-1;
      while(k1<k2){
            p1=p[k2]; p[k2]=p[k1]; p[k1]=p1;
            k1++; k2--;
      }
}
/*основная функция генерирования перестановок в лексикографическом по-
рядке*/
void perm(n){
      int j,p1;
      if(n==1)write();....
      else{
            for(j=0; j<n; j++){
                  perm(n-1); //рекурсивный вызов
                  p1=p[n-1]; p[n-1]=p[j]; p[j]=p1;
                  reverse(n-1);
            }
      }
}
/*сортировка массива p*/
void sort(int n) {
      int k1,k2,p1,g;
      for(k1=n-1; k1>=0; k1--){
            g=0;
            for(k2=0; k2<k1; k2++){
                  if(p[k2]>p[k2+1]){
                        p1=p[k2];
                        p[k2]=p[k2+1];
                        p[k2+1]=p1;
                        g=1;
                  }
            }
            if(!g)break;
    }
```

```
}
//
int main() {
      int a,i=0;
      while(scanf("%d",&a)!=-1){
            p[i]=a;
            i++;
            if(i==20)break;
      }
      m=i;
      sort(i);
      perm(i);
      return 0;
}
```

Комментарий к листингу 15.

Для того, чтобы понять, как программа из листинга 15 генерирует перестановки следует более подробно остановиться на некоторых понятиях. Наш алгоритм использует понятие так называемого лексикографического порядка.

Листинг 16. Примеры лексикографического (а) и антилексикографического (б) порядков

```
(а)          (б)
1 2 3        1 2 3
1 3 2        2 1 3
2 1 3        1 3 2
2 3 1        3 1 2
3 1 2        2 3 1
3 2 1        3 2 1
```

Зная как сгенерировать последовательность, которая подчиняется по отношении текущей перестановке требованиям лексикографического или антилексикографического порядков, мы сможем построить нужный нам алгоритм. Дадим теперь определения лексикографического и антилексикографического порядков.

Пусть имеется две перестановки $(x_1,x_2,\ldots x_n)$ и $(y_1,y_2,\ldots y_n)$. Будем считать, что они лексикографически упорядочены, если существует номер k, что $x_k<y_k$ и $x_l=y_l$, для всех $l<k$. Для антилексикографического упорядочивания имеем: существует номер k, такой, что $x_k>y_k$ и $x_l=y_l$ для всех $l > k$. В нашем алгоритме мы используем именно антилексикографическое упорядочение, которое дает чуть менее сложный код. Обратитесь к листингу 16, и убедитесь, что данные определения подходят к представленным в листинге примерам.

Примем как данность следующие два важных (доказываемых математически) утверждения:

1. Первый элемент в последовательности упорядоченных множеств должен начинаться с отсортированного массива (множества).
2. Для генерации следующего массива нужно произвести такие действия:
    a. Поменять местами j и i элемент. Вначале j равно номеру первого элемента, а i последнего. Затем на каждом шаге j увеличивается, а i уменьшается на 1.

b. Произвести реверсию (см. функцию reverse в листинге 15) первых n-1 элементов множества. Потом будет реверсия n-2 элементов, реверсия n-3 и т.д.

На этом и основывается весь, в общем-то, не столь сложный алгоритм. Поскольку на входе мы можем получить и неупорядоченный массив, мы в начале сортируем его (см. функция sort). Так как входное множество заведомо не велико (слишком много перестановок, даже для небольшого n), то для упорядочения его используется простой алгоритм пузырьковой сортировки (функция sort).

Задания

1. Переведите программу из листинга 15 на язык программирования Pascal.
2. Попробуйте переписать программу для случая использования лексикографического упорядочивания.