Единственной женщине посвящаю.

Предисловие

Курс «Программирование в алгоритмах» является естественным продолжением курса «Основы программирования». Его содержание достаточно традиционно: структурное программирование, технологии «сверху — вниз» и «снизу — вверх». Освоению обязательной программы курса автор придает огромное значение. Она обязана стать естественной схемой решения задач, если хотите — культурой мышления или познания.

Только после этого, по нашему глубокому убеждению, разумно переходить на объектно-ориентированное программирование, работу в визуальных средах, на машинно-ориентированное программирование и т. д. Практика подтвердила жизненность данной схемы изучения информатики. Ученики физико-математического лицея г. Кирова многие годы представляют регион на различных соревнованиях по информатике, включая Международные олимпиады. Возвращение их без дипломов или медалей — редкое исключение. Переходя в разряд студентов, выпускники лицея без особых хлопот изучают дисциплины по информатике в любом высшем учебном заведении России, а также успешно выступают в командных чемпионатах мира среди студентов по программированию.

Для кого предназначен учебник? Во-первых, для учителей и учащихся школ с углубленным изучением информатики. Во-вторых, для студентов высших учебных заведений, изучающих программирование и стремящихся достичь профессионального уровня. Особенно он будет полезен тем, кто готовится принять участие в олимпиадах по программированию, включая широко известный чемпионат мира по программированию, проводимый под эгидой международной организации ACM (Association for Computing Machinery).

О благодарностях, ибо не так часто вслух предоставляется возможность сказать коллегам о том, что ты их безмерно уважаешь. Во-первых, это касается моих учеников и учителей одновременно. Без сотрудничества с ними вряд ли автор смог написать что-то подобное. Первая попытка написания книг такого рода была предпринята в 1993 году с Антоном Валерьевичем Лапуновым. То было совместное вхождение в данную проблематику. Для Антона более легкое, для автора достаточно

трудное. Затем последовало сотрудничество с Виталием Игоревичем Беровым. Наверное, благодаря ему автор нашел ту схему изложения алгоритмов на графах, которую он затем применял в работе даже с восьмиклассниками. Сотрудничество с Виктором Александровичем Матюхиным в пору его ученичества было не таким явным, но после того, как он стал студентом МГУ и в настоящее время, оно, на мой взгляд, очень плодотворно. Влияние Виктора на развитие олимпиадной информатики в г. Кирове просто огромно. С братьями Пестовыми, Андреем и Олегом, написаны книги. Более трудолюбивых и отзывчивых учеников автору не приходилось встречать. Сотрудничество с такими ребятами не было бы возможным без высочайшего профессионализма Владислава Владимировича Юферева и Галины Константиновны Корякиной, директора и завуча физико-математического лицея г. Кирова. Особые слова признательности хотелось бы выразить Владимиру Михайловичу Кирюхину, руководителю сборной команды школьников России по информатике. В 1994 году он привлек автора к работе в жюри российской олимпиады школьников. За эти годы автор воочию увидел весь тот тяжелейший труд, который стоит за победами российских школьников на Международных олимпиадах. Иосиф Владимирович Романовский сделал ряд ценных замечаний по газетной версии главы 5, за что автор благодарит его и желает дальнейших творческих успехов в деле обучения петербургских студентов. Многие годы главный редактор газеты «Информатика» Сергей Львович Островский поддерживал автора в его работе, и признательность за это остается неизменной.

Об ошибках. Учебники такого плана не могут не содержать ошибок. Для многих алгоритмов можно, естественно, найти другие схемы реализации. Автор с признательностью примет все замечания. Присылайте их, пожалуйста, по адресу okulov@vspu.kirov.ru

1. Арифметика многоразрядных целых чисел

Известно, что арифметические действия, выполняемые компьютером в ограниченном числе разрядов, не всегда позволяют получить точный результат. Более того, мы ограничены размером (величиной) чисел, с которыми можем работать. А если нам необходимо выполнить арифметические действия над очень большими числами, например
30!= 265252859812191058636308480000000?
В таких случаях мы сами должны позаботиться о представлении чисел и о точном выполнении арифметических операций над ними.

1.1. Основные арифметические операции

Числа, для представления которых в стандартных компьютерных типах данных не хватает количества двоичных разрядов, называются иногда «длинными». В этом случае программисту приходится самостоятельно создавать подпрограммы выполнения арифметических операций. Рассмотрим один из возможных способов их реализации.

Представим в виде:
$30!=2\times(10^4)^8+6525\times(10^4)^7+2859\times(10^4)^6+8121\times(10^4)^5+9105(10^4)^4+$
$+8636\times(10^4)^3+3084\times(10^4)^2+8000\times(10^4)^1+0000\times(10^4)^0$.
Это представление наталкивает на мысль о массиве.

Номер элемента в массиве A	0	1	2	3	4	5	6	7	8	9
Значение	9	0	8000	3084	8636	9105	8121	2859	6525	2

Возникают вопросы. Что за 9 в $A[0]$, почему число хранится «задом наперед»? Ответы очевидны, но подождем с ними. Будет ясно из текста.

Примечание

Мы работаем с положительными числами!

Первая задача. Ввести число из файла.
Но прежде описание данных.

```
Const MaxDig=1000;{*Максимальное количество цифр –
четырехзначных.*}
Osn=10000;{*Основание нашей системы счисления, в
элементах массива храним четырехзначные числа.*}
Type TLong=Array[0..MaxDig] Of Integer;{*Вычислите
максимальное количество десятичных цифр в нашем числе.*}
```

Прежде чем рассмотреть процедуру ввода, приведем пример. Пусть в файле записано число 23851674 и основанием (*Osn*) является 1000 (храним по три цифры в элементе массива *A*). Изменение значений элементов массива *A* в процессе ввода отражено в таблице 1.1. В основу положен посимвольный ввод, для этого используется переменная *ch*.

Таблица 1.1

A[0]	*A*[1]	*A*[2]	*A*[3]	*ch*	Примечание
3	674	851	23	-	Конечное состояние
0	0	0	0	2	Начальное состояние
1	2	0	0	3	1-й шаг
1	23	0	0	8	2-й шаг
1	238	0	0	5	3-й шаг
2	385	2	0	1	4-й шаг
2	851	23	0	6	5-й шаг
2	516	238	0	7	6-й шаг
3	167	385	2	4	7-й шаг
3	674	851	23		

Итак, в *A*[0] храним количество задействованных (ненулевых) элементов массива *A* — это уже очевидно. И при обработке каждой очередной цифры входного числа старшая цифра элемента массива с номером *i* становится младшей цифрой числа в элементе *i*+1, а вводимая цифра будет младшей цифрой числа из *A*[1].

Примечание (методическое)

Можно ограничиться этим объяснением и разработку процедуры вынести на самостоятельное задание. Можно продолжить объяснение. Например, выписать фрагмент текста процедуры переноса старшей цифры из *A*[*i*] в младшую цифру *A*[*i*+1], т. е. сдвиг уже введенной части на одну позицию вправо:

```
For i:=A[0] DownTo 1 Do
  Begin
    A[i+1]:=A[i+1]+(LongInt(A[i])*10) Div Osn;
    A[i]:=(LongInt(A[i])*10) Mod Osn;
  End;
```

Пусть мы вводим число 23851674 и первые 6 цифр уже разместили «задом наперед» в массиве *A*. В символьную переменную *ch* считали очередную цифру многоразрядного числа — это «7». По нашему алгоритму она должна быть размещена как младшая цифра в *A*[1]. Выписанный фрагмент программы освобождает место для этой цифры. В таблице 1.2 отражены результаты работы этого фрагмента.

Таблица 1.2

i	*A*[1]	*A*[2]	*A*[3]	*ch*
2	516	238	0	7
2	516	380	2	
1	160	385	2	

После этого остается только добавить текущую цифру к *A*[1] и изменить значение *A*[0].

В конечном итоге процедура должна иметь следующий вид:

```
Procedure ReadLong(Var A:TLong);
 Var ch:Char;i:Integer;
 Begin
  FillChar(A,SizeOf(A),0);
  Repeat
   Read(ch);
  Until ch In ['0'..'9']
              {*Пропуск не цифр в начале файла.*}
  While ch In ['0'..'9'] Do
    Begin
      For i:=A[0] DownTo 1 Do
        Begin{*"Протаскивание"
              старшей цифры в числе из A[i] в младшую
              цифру числа из A[i+1].*}
          A[i+1]:=A[i+1]+(LongInt(A[i])*10) Div Osn;
          A[i]:=(LongInt(A[i])*10) Mod Osn;
        End;
```

```
      A[1]:=A[1]+Ord(ch)-Ord('0');
      {*Добавляем младшую цифру к числу из A[1].*}
     If A[A[0]+1]>0 Then Inc(A[0]);
        {*Изменяем длину, число задействованных
         элементов массива A.*}
     Read(ch);
   End;
 End;
```

Вторая задача. Вывод многоразрядного числа в файл или на экран.

Казалось бы, нет проблем — выводи число за числом. Однако в силу выбранного нами представления числа необходимо всегда помнить, что в каждом элементе массива хранится не последовательность цифр числа, а значение числа, записанного этими цифрами. Пусть в элементах массива хранятся четырехзначные числа. И есть число, например, 128400583274. При выводе нам необходимо вывести не 58, а 0058, иначе будет потеря цифр. Итак, нули также необходимо выводить. Процедура вывода имеет вид:

```
Procedure WriteLong(Const A:TLong);
 Var ls,s:String;
     i:Integer;
 Begin
  Str(Osn Div 10,ls);
  Write(A[A[0]]);{*Выводим старшие цифры числа.*}
  For i:=A[0]-1 DownTo 1 Do
    Begin
      Str(A[i],s);
      While Length(s)<Length(ls) Do s:='0'+s;
        {*Дополняем незначащими нулями.*}
      Write(s);
    End;
  WriteLn;
 End;
```

Третья задача. Сложение двух положительных чисел.

Предварительная работа по описанию способа хранения, вводу и выводу многоразрядных чисел выполнена. У нас есть все необходимые «кирпичики», например, для написания программы сложения двух положительных чисел. Исходные числа и результат храним в файлах. Назовем процедуру сложения *SumLongTwo*. Тогда программа ввода двух чисел и вывода результата их сложения будет иметь следующий вид:

```
Var A,B,C:TLong;
Begin
 Assign(Input,'Input.txt'); Reset(Input);
 ReadLong(A);ReadLong(B);
 Close(Input);
 SumLongTwo(A,B,C);
 Assign(Output,'Output.txt'); Rewrite(Output);
 WriteLong(C);
 Close(Output);
End.
```

Трудно ли объяснить процедуру сложения? Используя простой пример — нет.

Пусть A=870613029451, B=3475912100517461.

Результат — C=3476782713546912. Алгоритм имитирует привычное сложение столбиком, начиная с младших разрядов. И именно для простоты реализации арифметических операций над многоразрядными числами используется машинное представление «задом наперед». Ниже приведен текст процедуры сложения двух чисел.

```
Procedure SumLongTwo(Const A,B:TLong;Var C:TLong);
 Var i,k:Integer;
 Begin
  FillChar(C,SizeOf(C),0);
  If A[0]>B[0] Then k:=A[0] Else k:=B[0];
  For i:=1 To k Do
    Begin
     C[i+1]:=(C[i]+A[i]+B[i])Div Osn;
     C[i]:=(C[i]+A[i]+B[i]) Mod Osn;
     {*Есть ли в этих операторах ошибка?*}
    End;
  If C[k+1]=0 Then C[0]:=k Else C[0]:=k+1;
 End;
```

Таблица 1.3

i	*A*[i]	*B*[i]	*C*[1]	*C*[2]	*C*[3]	*C*[4]
1	9451	7461	6912	1	0	0
2	1302	51	6912	1354	0	0
3	8706	9121	6912	1354	7827	1
4	0	3475	6912	1354	7827	3476

Четвертая задача. Реализация операций сравнения чисел: $A=B$, $A<B$, $A>B$, $A\le B$, $A\ge B$.

Функция $A=B$ имеет вид.

```
Function Eq(Const A,B:TLong):Boolean;
Var i:Integer;
Begin
  Eq:=False;
  If A[0]=B[0] Then
    Begin
      i:=1;
      While (i<=A[0]) And (A[i]=B[i]) Do Inc(i);
      Eq:=(i=A[0]+1);
    End;
End;
```

Реализация функции $A>B$ также прозрачна.

```
Function More(A,B:TLong):Boolean;
 Var i:Integer;
 Begin
  If A[0]<B[0]
    Then More:=False
    Else
      If A[0]>B[0]
        Then More:=True
        Else
          Begin
            i:=A[0];
            While (i>0) And (A[i]=B[i]) Do Dec(i);
            If i=0
              Then More:=False
              Else
                If A[i]>B[i]
                  Then More:=True
                  Else More:=False;
          End;
 End;
```

Остальные функции реализуются через функции *Eq* и *More*.

```
Function Less(A,B:TLong):Boolean;{A<B}
 Begin
  Less:=Not(More(A,B) Or Eq(A,B));
 End;
```

```
Function More_Eq(A,B:TLong):Boolean;
 Begin
  More_Eq:=More(A,B) Or Eq(A,B);
 End;
```

И наконец, последняя функция $A \le B$.

```
Function Less_Eq(A,B:TLong):Boolean;
 Begin
  Less_Eq:=Not(More(A,B));
 End;
```

Для самостоятельного решения может быть предложена следующая более сложная задача. Требуется разработать функцию, которая выдает 0, если A больше B, 1, если A меньше B и 2 при равенстве чисел. Но сравнение должно быть выполнено с учетом сдвига. О чем идет речь? Поясним на примере.

Пусть A равно 56784, а B — 634. При сдвиге на 2 позиции влево числа B функция должна сказать, что B больше A, без сдвига — что A больше B. Или A равно 567000, а B — 567 и сдвиг равен 3, то функция должна «сказать», что числа равны.

Решение может иметь следующий вид.

```
Function More(Const A, B: TLong; sdvig: Integer):
  Byte;
 Var i: Integer;
  Begin
   If A[0]>(B[0]+sdvig)
    Then More:=0
    Else
     If A[0]<(B[0]+sdvig)
      Then More:=1
      Else
       Begin
        i:=A[0];
        While (i>sdvig) And (A[i]=B[i-sdvig]) Do Dec(i);
        If i=sdvig
         Then
          Begin
           More:=0;{*Совпадение чисел с учетом
                    сдвига.*}
          For i:=1 To sdvig Do
           If A[i]>0 Then Exit;
              More:=2;{*Числа равны, "хвост" числа A
                       равен нулю.*}
```

```
        End
       Else More:=Byte(A[i]<B[i-sdvig]);
     End;
  End;
```

Пятая задача. Умножение многоразрядного числа на короткое.

Под коротким числом понимается целое число, не превосходящее основание системы счисления.

Процедура очень похожа на процедуру сложения двух чисел.

```
Procedure Mul(Const A: TLong;Const K: LongInt;
              Var C: TLong);
  Var i: Integer;
{*Результат - значение переменной C.*}
  Begin
   FillChar(C, SizeOf(C), 0);
   If K=0
    Then Inc(C[0]){*Умножение на ноль.*}
    Else
      Begin
       For i:=1 To A[0] Do
        Begin
         C[i+1]:=(LongInt(A[i])*K+C[i]) Div Osn;
         C[i]:=(LongInt(A[i])*K +C[i]) Mod Osn;
        End;
        If C[A[0]+1]>0
          Then C[0]:=A[0]+1
          Else C[0]:=A[0];
           {*Определяем длину результата.*}
      End;
  End;
```

В качестве *самостоятельной работы* можно предложить разработку процедуры умножения двух чисел. Возможный вариант процедуры приведен ниже.

```
Procedure MulLong(Const A, B: TLong; Var C: TLong);
{*Умножение "длинного" на "длинное".*}
 Var i, j: Word;
       dv: LongInt;
 Begin
  FillChar(C, SizeOf(C), 0);
  For i:=1 To A[0] Do
```

[. . .]