Замечание. Выяснить, что напечатает функция для неположительных значений nx, ny.

1.39. Реализовать консольный калькулятор для целых чисел на основе набора функций так, чтобы функция `main()` отвечала только за ввод чисел и кода операции, вызов соответствующей функции и печать результата.

Правдивая история
— *Так что же делает Ваша программа?*
— *Работает!!*
— *А точнее?*
— *Считает!*
— *Что считает?*
— *Числа.*
— *Какие числа?*
— *Типа double, кажется ...*

1.2 Задачи на обработку последовательности

Общая постановка задач этого раздела выглядит следующим образом: имеется некоторое количество однотипных данных (например, чисел), и требуется вычислить некоторую характеристику этого набора (функцию от этих данных). Специфика задачи состоит в том, что мы не можем сохранить весь набор данных в памяти, поскольку обычно нам заранее неизвестно общее количество элементов. Поэтому нужно построить алгоритм, вычисляющий необходимую характеристику за один проход (просмотр) последовательности. Допускается сохранение и пересчет лишь конечного набора промежуточных значений, на основе которых в любой момент можно вычислить требуемую характеристику.

На практике алгоритмы решения подобных задач сводятся к построению некоторых рекуррентных соотношений между искомыми значениями, а также, возможно, и другими промежуточными результатами вычислений, чтобы можно было их пересчитывать, продвигаясь шаг за шагом от начала к концу последовательности

Подробное изложение вопросов, связанных с возможностью построения подобных алгоритмов, см. в [1].

1.40. С клавиатуры вводится положительное целое число n, а далее — последовательность, состоящая ровно из n целых чисел. Требуется найти и распечатать сумму элементов последовательности.
Решение.

```
#include <stdio.h>
int  main (void)
{
  double  x, sum;
  int  i, n;
  printf("  ----Вычисление суммы действительных чисел---- \n");
```

```
  printf("Введите число элементов последовательности n\n");
  scanf("%d",&n);
  if( n<=0 ) { printf("Последовательность пустая. \n"); return -1; }
  printf("Вводите элементы последовательности по одному \n");
  sum = 0.;
  for ( i=0; i<n; i++)
  {
    printf("Введите элемент x_[%d] последовательности \n", i );
    scanf( "%lf", &x );
    sum += x;
  }
  printf( "Последовательность содержит %d чисел \n", n );
  printf( "Их сумма равна %f\n", sum );
  return 0;
}
```

1.41. С экрана вводится непустая последовательность действительных чисел, длина которой заранее неизвестна. Ввод завершается либо комбинацией клавиш $Ctrl + d$, либо набором произвольных нечисловых данных. Требуется найти и распечатать произведение элементов последовательности.

Указание. Выделим непосредственное вычисление произведения в отдельную функцию

```
double Prod (void)
{
  int k;
  double x, p=1.;
  while(1)
  {
    k = scanf( "%lf", &x );
    if( k == 1) { p *= x; }
    else { break; }
  }
  return p;
}
```

Замечание. Процедура ввода данных с экрана является исключительно трудоемкой и утомительной (особенно на этапе тестирования программы). Поэтому далее будем считать, что обрабатываемая последовательность хранится в некотором текстовом файле, например, с именем `input.txt`. В простейшем варианте для работы с этими данными можно в момент запуска исполняемого файла `tsk.exe` с кодом нашей программы переопределить поток стандартного ввода, назначив источником данных файл `input.txt`. Это можно сделать следующей консольной командой `./tsk.exe < input.txt`. Однако такой способ является достаточно специфическим и частным, и более правильным было бы заставить программу работать непосредственно с требуемым файлом, выполняя стандартные процедуры открытия, чтения/записи и закрытия файла.

1.42. В файле `input.txt` задана последовательность действительных чисел $x_1, x_2, \ldots, x_n$ неизвестной длины (возможно пустая). Требуется найти

и распечатать значение $S = x_1 - x_2 + \ldots + (-1)^k x_k \ldots$, прочитав все числа, записанные в файле. Если последовательность пуста, то надо выдать соответствующее сообщение.

Приведем полное решение этой задачи как некий образец для дальнейшего.

Решение.

```
#include <stdio.h>
int Process (FILE *f, double *res); // прототип функции обработки файла
int  main (void)
{
  double  result;
  int retcode;
  FILE * f;
  f = fopen("input.txt","r");
  if ( f == NULL) { printf("Ошибка открытия файла\n"); return -1; }
  retcode = Process( f, &result );
  if ( retcode < 0 ) {
    printf("В файле нет данных \n");
  } else {
    printf("Результат: %f\n", result );
  }
  fclose(f);
  return retcode;
}
int Process (FILE *f, double *res)
{
   double s = 0, sign = 1, x;
   int key = -1, n;
   for ( n=0; fscanf( f, "%lf", &x ) == 1; n++ ) {
     s += sign*x;
     sign = -sign;
     key = 0;
  }
  *res = s;
  return key;
}
```

Замечание. Переменная `f` хранит *адрес* ячейки оперативной памяти, где размещен объект типа `FILE`, являющийся структурой (описание типа `FILE` имеется в `stdio.h`). Данная структура, создаваемая и заполняемая функцией `fopen`, содержит необходимую информацию для работы функции `fscanf`, т.е. для чтения данных из файла `input.txt`. Приведенный вызов

```
f=fopen("input.txt","r");
```

является компактной формой записи следующей конструкции:

```
const char * fname = "input.txt";
const char * fstr = "r";
f=fopen(fname,fstr);
```

Функция `fopen` принимает два указателя: на строку с именем открываемого файла и на строку с типом доступа. Если функция `fopen` присвоила переменной `f` значение именованной константы `NULL`, т.е. указателю на нулевую ячейку, то требуемый доступ к файлу обеспечить не удалось.

Функция `Process ()` подсчитывает количество `n` элементов последовательности (формально это значение не требуется, но указанный прием полезен в последующих задачах) и возвращает содержимое переменной `key`, значение которой равно нулю в случае успешного завершения, либо коду ошибки (в данном случае -1). Искомая величина `s` вычисляется внутри функции `Process()` и становится известной в `main()`, т.к. записывается по адресу, хранящемуся в `res`, и содержащему адрес ячейки `result`.

Следующий пример поясняет механизм обмена данными между функциями в языках C/C++.

```
#include<stdio.h>
void change_yes(int * x, int * y);
void change_no (int c, int d);
int main()
{
  int a=1,b=2;
  printf("Init: a=%d;b=%d\n",a,b);
   change_no(a,b);
  printf("change_no: a=%d;b=%d\n",a,b);
   change_yes(&a,&b);
  printf("change_yes: a=%d;b=%d\n",a,b);
return 0;
}
void change_no(int c, int d)
{       int tmp;
tmp=c; c=d; d=tmp;
return;
}
void change_yes(int *x, int *y)
{
  int c;
c=*x; *x=*y; *y=c;
return;
}
```

1.43. В чем идейная ошибочность следующих синтаксически верных конструкций?

```
void change_err1(int *x, int *y)
{
  int *c;
c=x; x=y; y=c;
return;
}
void change_err2(int *x, int *y)
{
```

```
  int *c;
*c=*x; *x=*y; *y=*c;
return;
}
```

1.44. Составьте программу для вычисления математического ожидания, т.е. среднего арифметического $M = \frac{1}{n}\sum_{i=1}^{n} x_i$ элементов числовой последовательности.

1.45. Вычислите дисперсию $D = \frac{1}{n}\sum_{i=1}^{n}(x_i - M)^2$ элементов числовой последовательности, т.е. среднее квадратичное отклонение от среднего арифметического.

Указание. Раскрыв скобки в требуемом выражении, можно заметить, что дисперсия явно выражается через сумму чисел, сумму их квадратов, и общее количество чисел, а эти три величины легко вычисляются по рекуррентным формулам за один просмотр последовательности: $D = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - M^2$.

1.46. Вычислите среднее геометрическое $G = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}$ элементов положительной числовой последовательности.

1.47. Вычислите среднее гармоническое $A_{-1} = n \Big/ \sum_{i=1}^{n} \frac{1}{x_i}$ элементов положительной числовой последовательности.

Для тестирования рассмотренных задач полезно подготовить набор тестовых файлов с числовыми последовательностями. В зависимости от сути задачи может потребоваться, чтобы эта последовательность удовлетворяла некоторые свойствам, например, была возрастающей, знакопеременной, являлась арифметической или геометрической прогрессией, и т.п. Одним из вариантов также является последовательность “случайных” чисел. Фактически нужно реализовать “генератор”, заполняющий файл числами, полученными по определенным формулам или правилам.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
// функции вычисления элементов последовательности
double NextValue (int i, double x);
int    RandValueI (int vmin, int vmax);
double RandValueD (double vmin, double vmax);
// функция заполнения файла
void   FillDataFile (FILE* f, int n);
int main(){
  char fname[256];   // имя выходного файла
  int n;              // количество чисел
  FILE* fout;
```

```
  printf ( "Введите имя файла" );
  scanf( ‘‘%s’’, fname );
  printf( " и количество элементов " );
  scanf( ‘‘%d’’, &n );
  fout = fopen( fname, ’’w’’ );
  if( fout == NULL ) { printf("Ошибка открытия файла %s\n", fname); return -1; }
  FillDataFile( fout, n );
  fclose(fout);
  return 0;
}
double NextValue ( int i, double x ) // детерменированная последовательность
{                                    // по явной рекуррентной формуле
  return x + i*i;
}
int    RandValueI ( int vmin, int vmax ) // случайное целое
                                         // из диапазона [vmin,vmax]
{
  int x = rand()%(vmax +1 - vmin) + vmin;      /*3*/
  return x;
}
int    RandValueD ( double vmin, double vmax ) // случайное вещественное
                                               // из диапазона
{
  double x = (double)rand()/(double)RAND_MAX;  /*3*/
  return x*(vmax - vmin) + vmin;
}
void   FillDataFile (FILE* f, int n)           /*1*/
{
  int i;
  double x, a=0, b=100;                        /*2*/
  srand(time(NULL));                           /*4*/
  for ( i=0; i<n; i++ ) {
    x = RandValueD ( a, b );
    fprintf( f, "%f\n", x );
  }
}
```

Замечания.

/*1*/. Выше приведены примеры трех функций для вычисления элемента последовательности и пример заполнения файла в случае использования функции `RandValueD`. Изменением соответствующих строк, не трудно построить примеры и для двух других функций.

/*2*/. Значения параметров для вычислений заданы здесь для простоты в явном виде. Можно их вынести в параметры процедуры и тем самым сделать ее более универсальной.

/*3*/. При каждом обращении функция `rand()` возвращает псевдослучайное число из диапазона от 0 до `RAND_MAX`, вычисленное по некоторой аналитической формуле. Таким образом, при каждом запуске программы будет получена одна и та же последовательность псевдослучайных чисел,

что полезно, если надо повторять тесты с одинаковыми наборами данных.

/*4*/. С помощью функции `srand(int)` можно сменить стартовую точку последовательности псевдослучайных чисел, например, сделав ее зависящей от текущего астрономического времени и тем самым практически всегда разной для разных запусков программы. В данном примере функция `time(NULL)` возвращает целое число, равное количеству секунд, прошедших с начала эры Юникса.

Идет контрольная работа тестового типа. Все старательно думают, а один бросает кубик и проставляет выпавший ответ. Все сдали свои работы, а он все бросает. – Вы чего не сдаете? – Да вот решил перепроверить ответы, а один все никак не сходится.

1.48. Подсчитайте количество положительных, отрицательных и нулевых чисел хранящейся в файле последовательности.
Решение.

```
#include <stdio.h>
int SignCount ( FILE *fin, int *neg, int *zero, int *pos );
int  main (void)
{
   int n=0, z=0, p=0;
   int res;
   char     fname [256];
   FILE    *fin;
   printf("Введите имя файла ->");
   scanf("%s",filename);
   fin = fopen (filename,"r");
   if ( !fin ) { printf("Ошибка открытия файла: %s\n",fname); return -1; }
   res = SignCount( fin, &m, &z ,&p );
   fclose(fin);
   if( res == 0 ){ printf("пустая последовательность\n"); return -1; }
   printf("Всего элементов %d\n из них\n", res );
   printf(" отрицательных %d\n", n );
   printf(" нулевых     %d\n", z );
   printf(" положительных %d\n", p );
   return 0;
}
int SignCount ( FILE *fin, int *neg, int *zero, int *pos )
{
   double  x;
   int n=0, z=0, p=0, cnt;
   for ( cnt=0; fscanf( fin, "%lf", &x) == 1; cnt++ ) {
     if ( x < 0 ) {
       n++;
     } else if ( x > 0 ) {
       p++;
     } else {
       z++;
     }
```

```
    }
    *neg = n;
    *zero = z;
    *pos = p;
    return cnt;
}
```

Замечание. С точки зрения вычислительных процедур более правильно не сравнивать вещественные числа на точное равенство/неравенство (`a==b` или `a!=b` в нотации языка С), а вычислять насколько эти числа отличаются друг от друга, например, `fabs(a-b) < eps`, т.е. проверять мала ли величина модуля разности этих чисел. Поэтому мы постарались избежать явного сравнения с нулем по типу `if ( x==0 ) z++`. Во всех последующих задачах также будет предполагаться, что операции сравнения на равенство с вещественными числами являются опасными и их следует избегать, а в необходимых случаях такое сравнение всегда понимается "с точностью ε", которая является параметром программы.

В последующих задачах по умолчанию считается, что входные данные являются целочисленными, а их количество позволяет корректно вычислить требуемый ответ.

1.49. Найдите количество четных и количество нечетных элементов последовательности.

1.50. Найдите количество четных элементов последовательности, значения которых больше десяти и не превосходят ста, а также их процент (с точностью до десятых долей) от общего числа элементов.

1.51. Найдите количество элементов последовательности кратных трем, количество кратных пяти и количество кратных семи.

1.52. Найдите количество элементов последовательности равных трем, количество равных пяти и количество равных семи.

1.53. Найдите количество нечетных элементов последовательности, значения которых больше нуля, а также порядковый номер последнего такого элемента.

1.54. Найдите количество элементов последовательности кратных трем, значения которых меньше ста, а также порядковый номер первого такого элемента.

1.55. Найдите количество элементов последовательности делящихся на пять и имеющих в записи более трех цифр, а также номер последнего такого элемента.

1.56. Найдите количество элементов последовательности заканчивающихся нулем, значения которых меньше тысячи, а также номер первого такого элемента.

1.57. Найдите количество элементов последовательности, для которых остаток от деления на 3 равен остатку от деления на 5, а также значение последнего такого элемента.

1.58. Найдите количество четных элементов последовательности, для которых остаток от деления на пять ровно на единицу отличается от остатка при делении на 7, а также значение последнего такого элемента.

1.59. Найдите количество нечетных элементов последовательности, для которых остаток от деления на пять не более чем на два отличается от остатка при делении на 7, а также номер последнего такого элемента.

1.60. Введите с клавиатуры число x и определите, сколько раз x встречается в последовательности.

1.61. Введите с клавиатуры число x и определите порядковый номер первого числа, равного x.

1.62. Введите с клавиатуры число x и определите порядковый номер последнего числа, равного x.

1.63. Определите значение минимального (или максимального) элемента последовательности.

1.64. Определите порядковый номер первого числа, равного максимуму из всех чисел последовательности.

1.65. Определите количество чисел, равных минимуму из всех чисел последовательности.

1.66. Определите номер первого и последнего минимального элемента последовательности.

1.67. Распечатайте значения и порядковые номера трех элементов последовательности, имеющих наибольшие по модулю значения.

1.68. Среди элементов последовательности найдите такую пару, что их порядковые номера отличаются не менее, чем на три, а их сумма среди подобных пар имеет максимальное значение.

Указание. На текущем k-ом шаге, $k \geq 4$, будем хранить в переменных $a1, a2, a3$ три последних введенных элемента последовательности, в переменной $amax$ — наибольший из первых $(k-3)$ элементов, а в $Smax$ — искомую сумму. При этом на четвертом шаге $amax$ равен первому элементу последовательности, а искомая сумма равна $S = amax + a3$. Очередной шаг алгоритма заключается в следующем — пересчитываем $amax$, т.е. сравниваем $amax$, $a1$ и изменяем $amax$ на наибольшую из величин $\{amax, a1\}$; переприсваиваем $a1 = a2, a2 = a3$; считываем очередной элемент в $a3$ и пересчитываем $Smax$, т.е. заменяем $Smax$ на наибольшую из величин $\{Smax, amax + a3\}$.

1.69. Определите величину наибольшего отклонения элементов последовательности от их среднего арифметического.

1.70. Определите, все ли элементы последовательности равны между собой.

Решение. Реализуем функцию, возвращающую следующие значения:

1 — последовательность постоянна,

-1 — последовательность содержит различные элементы,

0 — в файле недостаточно данных.

```
int   type (FILE *fin){
   int x,y;
   int key = 0;
   if ( fscanf(fin,"%d",&x)!=1 ) return key;
   key = 1;
   while ( fscanf(fin,"%d",&y)==1 ){
     if ( x!=y) key = -1;
     x = y;
   }
   return key;
}
```

1.71. Определите, является ли последовательность действительных чисел строго возрастающей, строго убывающей или ни той, ни другой.
Решение. Реализуем функцию, возвращающую следующие значения:
2 — последовательность является строго возрастающей,
1 — последовательность является строго убывающей,
0 — последовательность не является ни убывающей, ни возрастающей,
-1 — в файле недостаточно данных.

```
int   type (FILE *fin){
   double x,y;
   int key = -1;
   if ( fscanf(fin,"%lf%lf",&x,&y)!=2 ) return key;
     key=0;
     if ( y<x) key = 1;
     if ( y>x ) key =2;
     x=y;
   while ( fscanf(fin,"%lf",&y)==1 )
   {
      if ( y<x && key==2 ) key = 0;
      if ( y>x && key==1 ) key = 0;
      x = y;
   }
   return key;
}
```

1.72. Составьте программу для проверки является ли последовательность арифметической прогрессией.

1.73. Составьте программу для проверки является ли последовательность геометрической прогрессией.

1.74. Обозначим элементы последовательности через x_k, $k = 1, 2, \ldots$. Введите с клавиатуры четыре числа a, b, c, d и проверьте, удовлетворяют ли элементы последовательности рекуррентному соотношению $ax_k + bx_{k+1} + cx_{k+2} = d$.

1.75. Определите количество чисел в последовательности, которые больше предшествующего числа.

1.76. Пусть последовательность является неубывающей. Определите количество различных элементов этой последовательности.

1.77. Пусть последовательность является неубывающей. Определите количество элементов, которые появляются в этой последовательности более n раз. Значение n вводится с клавиатуры.

1.78. Определите, сколько раз во входной последовательности встречается подпоследовательность $1, 2, 3, \ldots, 100$.

1.79. Определите длину наибольшего постоянного участка, т.е. максимальное количество подряд идущих элементов с одним и тем же значением.

1.80. Введите с клавиатуры число n и определите количество постоянных участков последовательности, имеющих длину не меньше n.

1.81. Определите длину постоянного участка с наибольшей суммой элементов.

1.82. Определите общее количество элементов в постоянных участках целочисленной последовательности.

1.83. Определите сколько элементов последовательности не содержится в постоянных участках длины 2 и более.

1.84. Составьте программу для вычисления математического ожидания элементов числовой последовательности, учитывая из каждого постоянного участка только один элемент.

1.85. Определите длину возрастающего участка последовательности с наибольшим числом элементов.

1.86. Определите длину возрастающего участка последовательности с наибольшей суммой.

1.87. Введите с клавиатуры число n и определите количество невозрастающих участков последовательности, имеющих длину не меньше n.

1.88. Найдите сумму четных элементов во всех возрастающих участках целочисленной последовательности.

1.89. Определите каких участков в последовательности больше — возрастающих, или невозрастающих.

Элемент последовательности назовем локальным максимумом (минимумом), если он *строго* больше (меньше) своих соседей. Элемент назовем локальным экстремумом, если он является либо локальным минимумом, либо локальным максимумом. Для упрощения допустимо считать, что первый и последний элементы не могут быть локальными экстремумами.

1.90. Определите наибольшее расстояние между локальными максимумами (минимумами) элементов последовательности.

1.91. Найдите среднее арифметическое локальных экстремумов последовательности.

1.92. Определите и напечатайте локальные экстремумы с соседними точками. Каждая тройка печатается с новой строки.

1.93. Определите и напечатайте все отрезки возрастания последовательности. Каждый участок печатается с новой строки.

1.94. Определите и напечатайте все отрезки монотонности последовательности с явным указанием типа монотонности. Каждый отрезок печатается с новой строки.

1.95. Последовательность чисел представляет собой коэффициенты многочлена, расположенные в порядке возрастания степеней. Введите с клавиатуры число x и вычислите значение многочлена и его производной в точке x.

Идеи реализации. Обозначим хранящиеся в файле коэффициенты через a_0, a_1,..., a_n. Тогда $P_n(x) = a_0 + a_1x^1 + \ldots + a_nx^n$ — значение многочлена в точке x, $D_k(x) = a_1 + 2a_2x + \ldots + na_nx^{n-1}$ — значение его производной. При этом переменную xk, содержащую величину x^k, удобно на очередном шаге перевычислять рекуррентно $xk = x * xk$.

1.96. Последовательность чисел представляет собой коэффициенты многочлена, расположенные в порядке убывания степеней. Введите с клавиатуры число x и вычислите значение многочлена и его производной в точке x.

Идеи реализации. Воспользуйтесь схемой Горнера вычисления многочлена и преобразуйте ее в рекуррентные соотношения, связывающее значения многочлена и его производной. Если обозначить a_k — k-й элемент последовательности коэффициентов, $P_k(x) = a_0x^k + a_1x^{k-1} + \ldots + a_k$ — значение многочлена k-й степени в точке x, $D_k(x) = ka_0x^{k-1} + (k-1)a_1x^{k-2} + \ldots + a_{k-1}$ — значение производной многочлена $P_k(x)$, то данные рекуррентные соотношения будут иметь вид

$$P_0(x) = a_0, \quad D_0(x) = 0,$$
$$D_k(x) = P_{k-1}(x) + xD_{k-1}(x), \quad P_k(x) = xP_{k-1}(x) + a_k, \quad k > 0.$$

1.97. Для последовательности $a_1, a_2, \ldots, a_N$ определите максимальную сумму подряд идущих элементов, т.е. найдите $\max\limits_{0 \le k_1 \le k_2 < N} \sum\limits_{i=k_1}^{k_2} a_i$.

Указание. Реализуйте алгоритм Д. Кадана — если текущая сумма элементов стала отрицательной, то выгоднее начинать новую подпоследовательность, чем продолжать старую.

1.98. Найдите границы k_1 и k_2 искомой в задаче 1.97 подпоследовательности.

1.99. Для последовательности $a_1, a_2, \ldots, a_N$ и заданных чисел $0 \le n_1 \le n_2 \le N$ определите максимальную сумму подряд идущих элементов при условии, что их количество k удовлетворяет условию $n_1 \le k \le n_2$, т.е. найдите величину

$$\max_{\substack{0 \le k_1 \le k_2 < N \\ n_1 \le k_2 - k_1 + 1 \le n_2}} \sum_{i=k_1}^{k_2} a_i.$$

Идеи реализации. Алгоритм решения удобно сформулировать на основе вспомогательного объекта $s[]$: $s[0] = 0$, $s[i] = \sum_{j=1}^{i} a[j]$. При этом для реализации содержимое массива $s[]$ хранить не требуется.

1.100. Найдите наибольшее произведение двух элементов последовательности с разными номерами, делящееся на 51.

1.101. Последовательность представляет собой координаты точек на плоскости, т.е. пары чисел (x_i, y_i), $i = 1, 2, \ldots$. Найдите координаты вершин прямоугольника с минимальной площадью и сторонами, параллельными осям координат, содержащего все точки.

1.102. Последовательность представляет собой координаты точек на плоскости, т.е. пары чисел (x_i, y_i), $i = 1, 2, \ldots$. Вычислите сколько можно образовать таких отрезков с концами из этого множества, что ни один его конец не лежит на осях координат, и отрезок имеет ровно одну точку пересечения с осями.

1.3 Задачи на работу с массивами

В данном разделе собраны простейшие задачи обработки, анализа и преобразования данных, организованных в одномерный массив. Прежде чем формулировать условия задач, рассмотрим базовые конструкции, используемые для создания и заполнения массива.

В качестве первого примера приведем фрагмент программы, сохраняющей в массиве первые сто чисел Фибоначчи, занумерованные для удобства с нуля: $x_0 = x_1 = 1$, $x_k = x_{k-1} + x_{k-2}$, $k > 2$. Будем считать, что эти числа представлены типом `long int`.

```
#define  NMAX   100
long int  x [NMAX];
     x[0]=x[1]=1;
      for (i=2;i<NMAX;i++){
         x[i] = x[i-1]+x[i-2];
      }
```

Другим вариантом может быть заполнение массива значениями, читаемыми из файла. Создадим массив некоторой заведомо достаточной длины, а при чтении будем контролировать реально получающееся количество элементов. Соответствующий фрагмент программы может выглядеть, например, так (рассматриваем массив вещественных чисел и предполагаем, что файл уже открыт и определяется указателем `fin`).

```
#define  NMAX   1000
double   mas [NMAX];
int      n;
for (n=0; n<NMAX; n++) if(fscanf(fin,"%lf",&mas[n])!=1) break;
```

По окончании цикла переменная `n` содержит фактическую длину массива.

Отметим, что максимальный размер, созданных таким образом локальных статических массивов существенно меньше размера доступной оперативной памяти, т.к. они последовательно помещаются в программном стеке заранее ограниченной глубины. Поэтому более эффективным является следующий подход. Будем считать, что первое число, записанное в файле с