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

данными есть длина массива. В этом случае создание массива можно осуществить следующей последовательностью операторов.

```
#include<stdlib.h>    // содержит прототипы
                      // используемых функций malloc() и exit()
int main(){
 double  *mas=NULL;   // ячейка для хранения адреса,
                      // начиная с которого будет выделена память
int n, nmax=0, ncur;
int key;
key=fscanf(fin,"%d",&nmax);
if ( key!=1 || nmax<=0){
     exit(1);
}
mas = (double*)malloc(nmax*sizeof(double));
if (mas==NULL){ printf("Не удалось выделить %d байт\n'', nmax*sizeof(double));
     exit(1);
   }
for(n=0;n<nmax;n++){
   key=fscanf(fin,"%lf",&mas[i])
   if( key!=1) break;
}
ncur=n; //число заполненных элементов
.....
free(mas);
```

В случае успеха в перемнную `mas` будет записан адрес ячейки памяти, начиная с которого выделено запрашиваемое количество байт. Соответствующее число будет сохранено перед этим блоком, что позволит корректно отработать функции `free`.

Если после выполнения первой группы операторов окажется, что `nmax` не удалось прочитать из файла или соответствующее значение меньше либо равно нулю, то выполнение программы следует прекратить. Если в переменную `mas` функция `malloc` запишет значение `NULL`, т.е. адрес нулевой ячейки (в большинстве с-компилляторов константа `NULL` определена как `#define NULL ((void*) 0)`), то это означает, что выделить необходимую память не удалось и, следовательно, дальнейшее выполнение программы невозможно. Формально подобные проверки всегда следует предусматривать в программном коде. Отметим, что на данный момент в большинстве linux-систем процесс выделения памяти функционирует существенно сложнее — доступная память считается в некотором смысле бесконечной и `malloc` всегда возвращает ненулевое значение. Однако, при попытке реально работать с выделенным массивом программа может быть аварийно завершена системой по причине нехватки памяти. По окончании цикла `for()` в переменную `ncur` записывется число реально считанных из файла значений, что в дальнейшем потребуется для работы с массивом. Здесь и далее мы считаем, что количество хранящихся в файле элементов массива может быть как меньше, так и больше переменной `nmax`.

1.103. Проведите эксперименты по динамическому выделению больших массивов в оперативной памяти.

Идеи реализации. Вызываем в цикле функцию `malloc(1024*1024)` и обнуляем элементы до тех пор, пока не исчерпаем ресурс. Проверьте, изменится ли картина при одновременном запуске десятка подобных задач, содержащих, для обеспечения временной задержки, вызовы функции

`sleep(unsigned int seconds)`

из unix-библиотеки `unistd.h`.

Величина $(mas + n)$ по определению равна адресу n–ой ячейки массива, а поэтому вычисляется по формуле $mas + n * sizeof(mas)$. Следовательно `*(mas+n)` эквивалентно записи `mas[n]`, т.е. означает содержимое n–ой ячейки. Следующий код заполняет массив из файла, а затем вызывает функцию `void prn_mas(double *, int)`, распечатывающую содержимое ячеек массива и их адресов на экране:

```
for(n=0;n<nmax;n++){
   fscanf(fin,"%lf",mas+n); //эквиваленто fscanf(fin,"%lf",&mas[n]);
}
prn_mas(mas,n); //эквиваленто prn_mas(&mas[0],n);
```

где

```
void prn_mas(double *m, int N){
int n;
for(n=0;n<N;n++){
 printf("n = <%25d>, Val(mas[n]) = <%25.18lf>,  Adr(mas[n]) = <%25p>\n",
         n, mas[n], mas+n);
         }
 return;
}
```

В данном случае в функцию `prn_mas` передается адрес нулевого элемента массива и количество элементов массива. Это позволяет не только распечатать, но и (при необходимости) изменить содержимое каждой ячейки `mas[i]`. Отметим, что заголовок функции печати можно заменить на формально эквивалентный `void prn_mas(double m[], int N)`.

Решения задач, сформулированных ниже, следует оформить в виде отдельной функции, которая получает в качестве параметров адрес начала массива (т.е. адрес его нулевого элемента), длину массива (а также другие значения, если это необходимо по условию) и без использования дополнительных массивов выполняет необходимые действия. В функции не разрешено ни считывать новые данные, ни печатать сообщения на экран. Все нештатные ситуации нужно обрабатывать через возвращаемые значения. Функция `main` должна заполнять массив числами из файла, определять его фактическую длину, вызвать соответствующую функцию и распечатывать результат ее работы.

Если в задаче массив преобразовывается с удалением элементов, то ответ (преобразованный массив) должен быть размещен в начале исходного

массива. При этом нужно вычислить количество элементов в новом полученном массиве. Если не сказано специально, то оставшиеся в конце и не входящие в ответ элементы исходного массива, можно “не чистить”, т.е. их значения могут оставаться любыми.

В задачах, где в условии упоминается несколько массивов, предполагается, что их длины позволяют разместить только исходные данные. Дополнительные массивы создавать нельзя.

На данном этапе вычислительная сложность алгоритмов не имеет принципиального значения, поэтому методы с квадратичной оценкой трудоемкости считаются вполне допустимыми. Вопросы о существовании эффективных алгоритмов остаются в качестве самостоятельных упражнений для любознательных.

Отметим также, что большинство предлагаемых задач решаются на основе однопроходных алгоритмов.

1.104. Определите симметричны ли значения элементов массива?
Решение. Приведем полное решение для случая, когда массив читается из файла и его длина определяется первым прочитанным числом.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int    symmetry (int *mas, int n);
int * make_vector_int(int n, const char *info_1, const char *info_2);
int    main (){
   int * mas=NULL;
   int      res,n,nmax=0;
   char     filename [256];
   FILE  * fin=NULL;
   printf("----Симметричность массива----\n Введите имя файла ->");
   scanf("%s",filename);
   fin = fopen (filename,"r");
   if (!fin) { printf("Не удается открыть файл %s\n",filename);
               return -1; }
   if ( fscanf(fin,"%d",&nmax)!=1 ){
      printf("Ошибка ввода длины последовательности\n");
      return -1;
   }
   if ( nmax<=0 ){
      printf("Ошибка ввода длины последовательности\n");
      return -1;
   }

   mas = make_vector_int(nmax, __FILE__, __FUNCTION__);

   for (n=0; n<nmax; n++)if(fscanf(fin,"%d",&mas[n])!=1)break;
   res = symmetry(mas,n);
   printf("%s\n",(res) ? "Да":"Нет");
   free(mas)
```

```
   fclose(fin);
   return 0;
}
int    symmetry (int * mas, int n){
   int i;
   for (i=0;i<n/2;i++){
       if ( mas[i]!=mas[n-i-1] ) return 0;
   }
   return 1;
}

int * make_vector_int(int n, const char *info_1, const char *info_2)
{
int *tmp = (int*)malloc(n * sizeof(int));
if (tmp == NULL) {
printf("Ошибка в  %s -> %s : не удалось выделить %d байт!\n",
info_1, info_2, n * sizeof(int));
exit(1);
}
memset(tmp, 0, n * sizeof(int));
return tmp;
}
```

1.105. Требуется переставить элементы массива в обратном порядке.
Решение.

```
void    invert (int *mas, int n){
   int i,tmp;
   for (i=0;i<n/2;i++){
     tmp=mas[i];
     mas[i]=mas[n-i-1];
     mas[n-i-1]=tmp;
    }
   return;
}
```

1.106. Требуется циклически сдвинуть элементы массива на одну позицию вправо.

1.107. Требуется циклически сдвинуть элементы массива на `k` позиций вправо с затратой `O(n)` действий (`n`-длина массива).
Решение. Ограничение `O(n)` действий не позволяет вызвать `k` раз функцию из задачи 1.106 Приведем решение с использованием функции из задачи 1.105.

```
void   move(int *mas, int n, int k){
   k%=n;
   invert (mas, n-k);
   invert (mas+n-k, k);
   invert (mas, n);
return;
}
```

Другой способ решения состоит в постановке элемента сразу на его новую позицию. При этом образуются цепочки элементов, которые требуется циклически сдвинуть. Количество таких цепочек равно наибольшему общему делителю чисел `n` и `k`.

1.108. Определите значение и индекс минимального элемента массива.
Решение. Поскольку здесь ответ составляют два числа, то будем получать индекс через возвращаемое значение функции, а значение минимума — через параметр-указатель. Приведем текст функции, обрабатывающей массив.

```
int   minimum (double *mas, int n, double *min){
   int i,m;
   *min=mas[0];
   m=0;
   for (i=1; i<n; i++)
       if ( *min<mas[i] ) { *min = mas[i]; m = i; }
   return m;
}
```

1.109. Определите какое число встречается в массиве наибольшее количество раз.

1.110. Введите с клавиатуры число x и определите индекс и значение элемента массива, ближайшего к числу x.

1.111. Введите с клавиатуры число x и определите к какому значению ближе всего x: к минимальному в массиве, максимальному или среднему арифметическому.

1.112. Введите с клавиатуры число x и определите количество элементов массива, расстояние от которых до x в два раза меньше, чем максимальное расстояние между x и элементами массива.

1.113. Каждый элемент массива (кроме первого и последнего) замените на полусумму соседних элементов.

1.114. Замените каждый элемент массива $a[i]$ на сумму исходных элементов от нулевого до i–го включительно.

1.115. Сгруппируйте положительные элементы массива в его начале, а отрицательные — в конце с сохранением их порядка.

Указание. Пока найдется такая пара индексов i_0 и j_0, что $a[i_0]$ — самый левый положительный элемент, $a[j_0]$ — самый левый отрицательный элемент и $i_0 < j_0$, то запоминаем $tmp = a[j_0]$, сдвигаем часть массива $a[i_0], \ldots, a[j_0 - 1]$ на одну позицию вправо и вставляем $a[i_0] = tmp$.

По сути алгоритм является оптимизированной пузырьковой сортировкой, учитывающей при сравнении только знак элементов массива.

1.116. Требуется сравнить два неупорядоченных целочисленных массива A и B как числовые множества без учета повторяющихся элементов, т.е. проверить выполнение условий: $A \subset B$, $B \subset A$, $A = B$.

1.117. По двум неупорядоченным целочисленным массивам построить третий, являющийся их объединением как числовых множеств, т.е. с удалением повторяющихся элементов. Найти его длину.

1.118. По двум неупорядоченным целочисленным массивам построить третий, являющийся их пересечением как числовых множеств. Найти его длину.

1.119. Удалите из массива все отрицательные элементы, а оставшиеся уплотните (т.е. сдвиньте к началу массива с сохранением их порядка).

1.120. Введите с клавиатуры число x и удалите из массива все элементы, превосходящие x, а оставшиеся уплотните.

1.121. Удалите из массива все одинаковые подряд идущие элементы, оставив только первый из них, а оставшиеся уплотните.

1.122. Удалите из массива все повторяющиеся элементы, оставив только первый такой элемент. Оставшиеся в массиве элементы уплотните.

1.123. Введите с клавиатуры число x и удалите из массива каждый элемент, делящийся нацело на x, а оставшиеся уплотните к началу массива.

В следующих задачах считается, что исходный массив имеет достаточную для требуемого дублирования его элементов длину.

1.124. Введите с клавиатуры число x и продублируйте каждый элемент массива, превосходящий x (т.е. вставьте рядом такой же элемент).

1.125. Требуется вставить между каждыми двумя положительными элементами исходного массива их среднее арифметическое.

1.126. Требуется заменить каждый отрицательный элемент исходного массива на три элемента, равных его абсолютной величине.

1.127. Назовем x-отрезком группу подряд идущих элементов массива, каждый из которых равен x. Для заданного числа x замените элементы каждого x-отрезка на полусумму элементов, прилегающих к этому отрезку справа и слева. Если x-отрезок расположен в начале или конце массива, то будем считать соответствующий крайний элемент равным нулю.

1.128. Назовем массив из N целых чисел счастливым, если существует такое $0 < k < N$, что сумма элементов с индексами от 0 до $k-1$ совпадает с суммой элементов с индексами от k до $N-1$. Определите является ли данный массив счастливым.

1.129. Назовем массив из целых чисел плотным, если множество значений элементов массива полностью заполняет некоторый отрезок $[a, b]$ (рассматриваются целые значения). Определите является ли данный массив плотным.

1.130. Получите массив биномиальных коэффициентов для степени N, последовательно вычисляя строки треугольника Паскаля. Для реализации можно использовать только один массив длины $N+1$.

1.131. Требуется элементы с четными индексами сгруппировать в начале массива с сохранением их исходного порядка относительно друг друга, а

элементы с нечетными индексами сгруппировать в конце массива также с сохранением их исходного порядка.

1.132. В массиве длины N требуется элементы с индексами $i \leq [(N+1)/2]$ переместить на позиции с четными индексами с сохранением их исходного порядка относительно друг друга, а оставшиеся элементы ($i > [(N+1)/2]$) разместить на позициях с нечетными индексами также с сохранением их исходного порядка. Т.е. начальная и конечная половины массива "перемешиваются" чередованием элементов.

1.133. Пусть в массиве последовательно записаны цифры некоторого длинного десятичного числа. Требуется реализовать функции "прибавляющие единицу" и "вычитающие единицу" из такого числа. При этом для реализации переноса из "старшего разряда" можно заранее запасти один лишний элемент в массиве.

1.134. Требуется реализовать функции сложения и вычитания двух длинных десятичных чисел, хранящихся в массивах в виде последовательности цифр.

В языках C/C++ имеется встроенный механизм работы с многомерными массивами. По определению, массивом называется одномерная совокупность его элементов. Следовательно, двумерным массивом является одномерный массив элементами которого являются одномерные массивы–строки, и поэтому двумерный массив в языках C/C++ в оперативной памяти хранится по строкам. Аналогично определяются трех-, четырех- и т.д. мерные массивы. При этом явных ограничений на максимально допустимую размерность не накладывается, однако, конкретная реализация компилятора такое ограничение обычно имеет.

1.135. В прямоугольной целочисленной таблице (матрице) из 100 строк и 10 столбцов, найти строку с максимальной суммой. Распечатать ее номер и значение суммы.

Решение.

```
int    maxsum_str(int  B[][10], int M, int *res);
int    sum_str(int *str, int N);
int main(){
  int A[100][10];
  int M=100,N=10;
  int m,n;
  int nmax, summax;

  for(m=0;m<M;m++){
   for(n=0;n<N;n++){
    A[m][n]=rand()%100;
   }
  }
  nmax = maxsum_str(A,M,&summax)
  printf("Строка с наибольшей суммой: номер = %d сумма = %d \n",nmax, summax);
return 0;
}
```

```
int    sum_str(int *str, int N){
 int n;
 int sum=0;
 for(n=0;n<N;n++) sum += str[n];
return sum;
}
int    maxsum_str(int  B[][10], int M){
  int m=0;
  int N=10;
  int s_max;
  int m_max;

 m_max=0;
 s_max=sum_str(&B[0][0],N);
 for(m=1;m<M;m++){
  s=sum_str(&B[m][0],N);
  if(s>s_max){
   s_max=s; m_max=m;
  }
 }
return m_max;
}
```

Замечание. Имя многомерного массива (как и в одномерном случае) соответствует адресу его начала и указывается при передаче в функцию в качестве параметра. Так как двумерный массив хранится по строкам, следовательно, на этапе компиляции для формирования кода обращения к ячейке `B[m][n]` необходимо знать длину строки — поэтому в прототипе и теле функции `maxsum_str()` соответствующее значение требуется указать явно. Данное ограничение значительно снижает гибкость получаемого кода. Отметим, что в функцию `sum_str()` передается адрес начальной ячейки очередной строки, поэтому дальнейшая работа с этой строкой не отличается от работы с обычным одномерным массивом.

1.136. Для целочисленной матрицы 10×20 найдите сумму максимальных элементов из каждой строки. Распечатайте значение суммы.

1.137. Дана матрица 10×10. Среди строк, имеющих более двух положительных элементов, найдите строку с максимальной суммой. Распечатайте ее номер и значение суммы.

1.138. Дана матрица 10×20. Переставьте её первую строку (если потребуется), со строкой, сумма модулей элементов которой максимальна. Распечатайте исходную и полученную матрицы.

1.139. Дана матрица 10×20. Переставьте (если потребуется) первый столбец, со столбцом, сумма квадратов элементов которого максимальна. Распечатайте исходную и полученную матрицы.

1.140. Дана квадратная матрица 20×20. Найдите строку сумма квадратов элементов которой максимальна, и переставьте ее со столбцом, имеющем такой же номер. Распечатайте исходную и полученную матрицы.

1.141. В двумерной матрице 10×20 найдите подматрицу с максимальной суммой элементов. Распечатайте исходную и найденную матрицы.

Указание. См. решение задач 1.97-1.98.

Напомним, что создаваемый указанным образом статический массив размещается в программном стеке, а поэтому его максимально допустимый размер существенно меньше размера доступной оперативной памяти. Для повышения гибкости программного кода работа с динамическими двумерными (многомерными) массивами обычно реализуется вручную на основе одномерного массива:

```
 int * cr_Mas(int M,int N);
 void prn_Mas(int * A, int M,int N);
int main(){
 int * A;
 int m,n,M=10,N=20;
 A = cr_Mas(M,N);
 if(A)prn_Mas(A,M,N);
 if(A)free(A);
return 0;
}

int * cr_Mas(int M,int N){
 const unsigned  int MaxNumber=100;
 int *Mas;
 int m,n;
 Mas=(int *)malloc(M*N*sizeof(int));
 if(Mas==NULL)return NULL;
 for(m=0;m<M;m++){
   for(n=0;n<N;n++){
    Mas[m*N+n]=rand()%MaxNumber;
   }
  }
return Mas;
}

void prn_Mas(int *A, int M,int N){
 for(m=0;m<M;m++){
   for(n=0;n<N;n++){
    fprintf(stdout,"%10d ", Mas[m*N+n]);
   }
  fprintf(stdout,"\n");
  }
return;
}
```

В некоторых случаях также уместна косвенная ссылочная реализация (см. раздел 2.5).