

# Основы организации данных и алгоритмизация (2012)

## Лекция 6. Простейшие структуры данных и алгоритмы

Штанюк А.А.

9 декабря 2012 г.



## 1 Простейшие структуры данных

- Введение
- Массивы
- Записи

## 2 Простейшие алгоритмы

- Введение
- Обмен

- Минимальный и максимальный элементы
- Инвертирование
- Группировка

## 3 Автоматный подход

## 4 Рекурсивные задачи

## 5 Вопросы для самоконтроля

## 6 Глоссарий

## 7 Библиографический список

## 1 Простейшие структуры данных

- Введение
- Массивы
- Записи

## 2 Простейшие алгоритмы

- Введение
- Обмен

- Минимальный и максимальный элементы
- Инвертирование
- Группировка

## 3 Автоматный подход

## 4 Рекурсивные задачи

## 5 Вопросы для самоконтроля

## 6 Глоссарий

## 7 Библиографический список

# Организация данных

Данные в программе должны быть организованы. Организация предполагает способ взаимосвязи между различными экземплярами данных.

О неорганизованности по отношению к данным можно говорить тогда, когда их экземпляры хранятся по различным адресам в памяти и никак не связаны между собой. Например, два целых числа, расположенных в памяти по адресу  $A_1$  и  $A_2$ .

Простейший способ организации - **массив**, когда однотипные данные располагаются в памяти строго последовательно, в смежных адресах.

arr[0] arr[1] arr[2] arr[3] arr[4] arr[5] arr[6]



A0 A1 A2 A3 A4 A5 A6

# Массивы

Для массива необходимо знать адрес первого элемента (базовый адрес) и количество элементов. Зная базовый адрес, можно вычислить смещение до любого элемента.

Достоинства массива:

- простота организации,
- высокая скорость доступа к элементам ( $O(1)$ ).

Недостатки:

- сложность операций добавления и удаления элементов,
- требуется непрерывный участок памяти.

# Записи

**Структура (запись)** представляет собой набор данных, хранящихся в памяти в смежных адресах, но не обязательно принадлежащих одному типу. Это позволяет рассматривать саму структуру как универсальный тип для представления внутреннего устройства множества объектов.

Скаляры фундаментальных типов, массивы и структуры можно отнести к *примитивным* способам организации данных, поскольку они требуют простейших операций для доступа к своим элементам.

Далее будут рассмотрены более сложные способы организации, требующие определенную взаимосвязь элементов и алгоритмы доступа к ним.

## 1 Простейшие структуры данных

- Введение
- Массивы
- Записи

## 2 Простейшие алгоритмы

- Введение
- Обмен

- Минимальный и максимальный элементы
- Инвертирование
- Группировка

## 3 Автоматный подход

## 4 Рекурсивные задачи

## 5 Вопросы для самоконтроля

## 6 Глоссарий

## 7 Библиографический список

# Введение

Алгоритмы на простейших структурах данных типа массива играют исключительно важную роль, поскольку используются чаще всего. Эти алгоритмы могут использоваться в более сложных структурах и АД.

Мы рассмотрим базовые алгоритмы применительно к простейшим типам (целые, вещественные числа), но результаты могут быть применены и к другим типам после прояснения смысла операции *сравнения* двух элементов.

Алгоритмы будем рассматривать на примере программ, написанных на языке Си++, без использования объектных возможностей.

## Простейшие операции

При построении и анализе алгоритмов простейшими будут называться следующие операции:

- Присвоение объекту  $A$  значения объекта  $B$  ( $A = B$ ).
- Сравнение значений двух объектов  
( $A == B, A != B, A > B, A < B, A >= B, A <= B$ ).
- Увеличение (уменьшение) значения объекта на константу  
( $A ++, B --, A+ = 5, B- = 10$ ).

### Постановка задачи

*Имеются два объекта (одинокные переменные или элементы массива).  
Требуется произвести обмен их значений.*

### Постановка задачи

*Имеются два объекта (одиночные переменные или элементы массива). Требуется произвести обмен их значений.*

### Замечание

Обмен будет реализован в виде процедуры (swap), которая будет использоваться во множестве других алгоритмов. В некоторых случаях для ускорения работы алгоритма обмен может быть реализован непосредственно

Реализация в классическом Си (с использованием указателей):

#### Функция

```
void swap(int *A, int *B)
{
    int T=*A;
    *A=*B;
    *B=T;
}
```

Функция `swap` получает два адреса: первого и второго элементов. С помощью операции *разыменования* функция получает доступ к содержимому изменяемых элементов.

Как можно воспользоваться процедурой `swap`?

```
int main()
{
    int x=10, y=20;
    printf("x=%d, y=%d\n",x,y); // 10,20
    swap(&x,&y);
    printf("x=%d, y=%d\n",x,y); // 20,10
    return 0;
}
```

### Реализация в C++ (с использованием ссылок):

#### Функция

```
void swap(int& A, int& B)
{
    int T=A;
    A=B;
    B=T;
}
```

В этой реализации используются *ссылки*, позволяющие напрямую обращаться к значениям переставляемых элементов.

# Минимальный и максимальный элементы

## Постановка задачи

*Имеется массив целых чисел размером  $N$  элементов. Найти максимальный (или минимальный) элемент*

# Минимальный и максимальный элементы

## Постановка задачи

*Имеется массив целых чисел размером  $N$  элементов. Найти максимальный (или минимальный) элемент*

## Замечание

Для поиска одного значения (макс. или мин.) можно использовать функцию, принимающую массив (адрес первого элемента) и размер массива в элементах, и возвращающую найденный элемент

## Минимальный и максимальный элементы

Внутренней переменной **max** присваиваем значение первого элемента массива, а потом сравниваем в цикле со всеми последующими. Если какой-то элемент оказывается больше чем **max**, то он становится максимальным и дальнейшие сравнения производятся с ним. После обработки всех элементов в **max** находится максимальное значение.

```
int getMax(int A[],int N)
{
    int max=A[0],i;
    for(i=1;i<N;i++)
        if(A[i]>max)
            max=A[i];
    return max;
}
```

## Минимальный и максимальный элементы

Здесь всё то же самое, только поиск производится по критерию минимального значения

```
int getMin(int A[],int N)
{
    int min=A[0],i;
    for(i=1;i<N;i++)
        if(A[i]<min)
            min=A[i];
    return min;
}
```

## Минимальное и максимальное значения

Задача немного усложняется, если требуется определить в одной функции и максимальное и минимальное значения. В этом случае **min,max** - это указатели, в которые при вызове функции заносятся адреса внешних переменных. При работе с указателями внутри функции мы должны использовать операцию **\***(разыменование) для доступа к значениям внешних переменных.

```
void getMinMax(int A[],int N, int *min, int *max)
{
    int i;
    *min=A[0],*max=A[0];
    for(i=1;i<N;i++) {
        if(A[i]<*min)
            *min=A[i];
        if(A[i]>*max)
            *max=A[i];
    }
}
```

А вот так можно воспользоваться **getMinMax**:

```
int main()
{
    int A[10]={6,3,8,5,1,3,9,4,7,2};
    int min,max;
    getMinMax(A,10,&min,&max);
    printf("Min=%d, Max=%d\n",min,max);
    return 0;
}
```

# Инвертирование

## Постановка задачи

*Имеется массив целых чисел размером  $N$  элементов. Переставить элементы в обратном порядке*

# Инвертирование

## Постановка задачи

*Имеется массив целых чисел размером  $N$  элементов. Переставить элементы в обратном порядке*

## Замечание

Суть инвертирования заключается в обратной перестановке элементов. В результате неё исходный массив преобразуется к новому виду. Задача решается без привлечения дополнительных массивов.

# Инвертирование

Для реализации алгоритма мы создаём две переменных,  $l, r$  для движения по массиву навстречу друг другу. Эти переменные содержат позиции переставляемых элементов. Как только позиции сравниваются, алгоритм заканчивает работу.

```
void Invert(int A[], int N)
{
    int l=0, r=N-1;
    while(l<r)
    {
        swap(&A[l], &A[r]);
        l++; r--;
    }
}
```

# Инвертирование

Обработка массива, основанная на принципе движения с двух концов навстречу друг другу является очень распространённым приёмом, используемым при построении множества алгоритмов.

Она позволяет достичь вычислительной сложности  $O(n)$

# Инвертирование

Пример: определение, является ли строка **палиндромом**

```
int ifPalindr(char *str)
{
    // настраиваем указатели на начало и конец строки
    char *beg=str,*end=str+strlen(str)-1;
    while(beg<end)
    {
        if(*beg!=*end)
            return 0;
        beg++; end--;
    }
    return 1;
}
```

Функция возвращает 1, если **да**, и 0 - если **нет**

### Постановка задачи

*Дан массив  $A$  и число  $B$ . Переставить элементы в массиве  $A$  таким образом, чтобы слева от некоторой границы стояли числа меньше  $B$ , а справа - больше  $B$ .*

# Группировка

## Постановка задачи

*Дан массив  $A$  и число  $B$ . Переставить элементы в массиве  $A$  таким образом, чтобы слева от некоторой границы стояли числа меньше  $B$ , а справа - больше  $B$ .*

## Замечание

Алгоритм группировки не предъявляет требований к следованию элементов внутри групп, поэтому сортировать все элементы - нерационально. Достаточно только переставлять элементы, двигаясь навстречу друг другу.

## Группировка

Массив просматривается с двух концов (аналогично алгоритму инвертирования). При нахождении нужной пары элементов выполняется их перестановка.

```
void Grouping(int A[],int N, int B)
{
    int l=0,r=N-1;
    while(l!=r)
    {
        if(A[l] <=B)
            l++;
        else if(A[r]>B)
            r--;
        else
        {
            swap(&A[l],&A[r]);
            l++; r--;
        }
    }
}
```

# Группировка

**Пример:**

Исходный массив:

5,8,3,4,4,7,5,0,3,8

Результат группировки при  $B = 4$ :

3,0,3,4,4,7,5,8,5,8

## 1 Простейшие структуры данных

- Введение
- Массивы
- Записи

## 2 Простейшие алгоритмы

- Введение
- Обмен

- Минимальный и максимальный элементы
- Инвертирование
- Группировка

## 3 Автоматный подход

## 4 Рекурсивные задачи

## 5 Вопросы для самоконтроля

## 6 Глоссарий

## 7 Библиографический список

# Автоматное программирование

Для многих задач программирования существуют эффективные решения, основанные на **автоматном** подходе.

Суть автоматного подхода заключается в том, что текущее состояние программы определяется значениями специальных переменных и при возникновении определённых событий, программа переходит в другое состояние (значения специальных событий меняются).

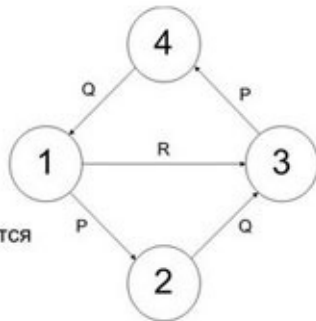
Автоматный подход широко применяется в задачах синтаксического анализа при построении парсеров, трансляторов и т.п.

# Автоматное программирование

Теоретически **автомат** представляется как матрица переходов, строками которой служат состояния автомата, а столбцами - входные символы).  
В качестве входных символов на практике можно рассматривать результаты проверки некоторых условий.

|   | a | b | c | d |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 1 |
| 2 | 3 | 4 | 1 | 2 |
| 3 | 4 | 1 | 2 | 3 |
| 4 | 1 | 2 | 3 | 4 |

Здесь состояния идентифицируются  
порядковыми номерами, а  
воздействия — буквами



# Автоматное программирование

Рассмотрим задачу:

## Задача

В числовом массиве определить количество последовательностей отрицательных чисел

## Решение

При движении по массиву мы определяем начало каждой последовательности и её окончание. При этом специальная переменная содержит одно из двух значений: 0 - мы находимся вне последовательности, 1 - внутри неё.

# Автоматное программирование

```
int getSeqNumber(int A[], int N)
{
    int inSeq=0, count=0, i;
    for(i=0; i<N; i++)
    {
        // если были вне последовательности и встретили отр.
        if(A[i]<0 && !inSeq)
        {
            inSeq=1; // последовательность началась
            count++; // увеличиваем счётчик
        }
        // если были в последовательности и встретили неотр.
        else if(A[i]>=0 && inSeq)
            inSeq=0;
    }
    return count; // возвращаем количество последовательностей
}
```

## 1 Простейшие структуры данных

- Введение
- Массивы
- Записи

## 2 Простейшие алгоритмы

- Введение
- Обмен

- Минимальный и максимальный элементы
- Инвертирование
- Группировка

## 3 Автоматный подход

## 4 Рекурсивные задачи

## 5 Вопросы для самоконтроля

## 6 Глоссарий

## 7 Библиографический список

# Понятие рекурсии

## Рекурсивный объект

Объект называется рекурсивным, если он содержит сам себя или определен с помощью самого себя.

## Рекурсивная процедура

Если процедура  $p$  содержит явное обращение к самой себе, то она называется явно рекурсивной. Если процедура  $p$  содержит обращение к некоторой процедуре  $q$ , которая в свою очередь содержит прямое или косвенное обращение к  $p$ , то  $p$  - называется косвенно рекурсивной.

В программировании глубина рекурсии ограничена и определяется размером стека. При каждом вызове функции стек увеличивается, а при каждом возвращении из рекурсивной функции стек уменьшается.

# Понятие рекурсии

Рекурсия в программировании:

## Прямая рекурсия

```
TYPE fun()  
{  
    ...  
    fun();  
}
```

## Косвенная рекурсия

```
TYPE fun1()  
{  
    ...  
    fun2();  
}  
TYPE fun2()  
{  
    ...  
    fun1();  
}
```

## Рекурсивные задачи

Рекурсию можно применять во всех случаях, когда в задаче можно выделить самоподобие. Например,

- 1 Сумма элементов массива равна сумме сумм элементов его подмассивов.
- 2 Нахождение максимального элемента массива является результатом сравнения первого элемента с максимальным элементом оставшейся части массива.

Использование рекурсии для решения задач не всегда эффективно с точки зрения исполнения, но всегда даёт выигрыш в лаконичности и красоте записи алгоритма

Существуют специальные задачи, решаемые обычно с помощью рекурсии:

- 1 Быстрая сортировка (QuickSort).
- 2 Вычисление факториала целого числа.
- 3 Вычисление чисел Фибоначчи.
- 4 Решение головоломки "Ханойские башни".
- 5 Генерация фракталов.
- 6 Обработка древовидных структур данных.

## О бесконечности и рекурсии

Мощь рекурсивного определения объекта в том, что такое конечное определение способно описывать бесконечно большое число объектов. С помощью рекурсивной программы же возможно описать бесконечное вычисление, причём без явных повторений частей программы.

Но рекурсивная программа не может вызывать себя бесконечно, иначе она никогда не остановится, таким образом в программе (функции) должна присутствовать еще один важный элемент - так называемое терминальное условие, то есть условие при котором программа прекращает рекурсивный процесс.

## Алгоритм построения рекурсивной функции

- Разработать заголовок рекурсивной функции
- Определить терминальное условие (условие возвращения)
- Определить связь текущей копии функции с результатом вызова себя самой



# Примеры

## Вычисление факториала

```
int factorial(int N)
{
    if(N<=1)
        return 1;
    else
        return N*factorial(N-1);
}
```

## Числа Фибоначчи

```
int fib(int N)
{
    if(N==1 || N==2)
        return 1;
    else
        return fib(N-1)+fib(N-2);
}
```

# Ханойские башни

## Задача

В одном из буддийских монастырей монахи уже тысячу лет занимаются перекладыванием колец. Они располагают тремя пирамидами, на которых надеты кольца разных размеров. В начальном состоянии 64 кольца были надеты на первую пирамиду и упорядочены по размеру. Монахи должны переложить все кольца с первой пирамиды на вторую, выполняя единственное условие - кольцо нельзя положить на кольцо меньшего размера. При перекладывании можно использовать все три пирамиды. Монахи перекладывают одно кольцо за одну секунду. Как только они закончат свою работу, наступит конец света.

## Решение

Нужно применить рекурсивно алгоритм, переложив  $n - 1$  кольцо с первой пирамиды на третью пирамиду. Затем сделать очевидный ход, переложив последнее самое большое кольцо с первой пирамиды на вторую. Затем снова применить рекурсию, переложив  $n - 1$  кольцо с третьей пирамиды на вторую пирамиду.

## Ханойские башни (реализация)

```
#include <stdio.h>

void han(int num,char a,char b,char c) {
    if(num>0)    {
        han(num-1,a,c,b);
        printf("%c--->%c\n",a,c);
        han(num-1,b,a,c);
    }
}

int main() {
    char a,b,c;
    int num;
    printf("Enter number of rings: ");
    scanf("%d",&num);

    a='A';b='B';c='C';
    if(num>0)
        han(num,a,b,c);
    else
        puts("Error!");
    return 0;
}
```

## 1 Простейшие структуры данных

- Введение
- Массивы
- Записи

## 2 Простейшие алгоритмы

- Введение
- Обмен

- Минимальный и максимальный элементы
- Инвертирование
- Группировка

## 3 Автоматный подход

## 4 Рекурсивные задачи

## 5 Вопросы для самоконтроля

## 6 Глоссарий

## 7 Библиографический список

## Вопросы для самоконтроля

- К каким способам организации данных относится массив? [Подсказка](#)
- В чем плюсы и минусы массивов? [Подсказка](#)
- Как определить значение произвольного элемента массива? [Подсказка](#)
- Что такое запись? [Подсказка](#)
- Чем запись отличается от массива? [Подсказка](#)
- Какие операции можно отнести к простейшим? [Подсказка](#)
- Как реализована в классическом Си процедура обмена значений двух областей памяти? [Подсказка](#)
- Как можно воспользоваться функцией обмена? [Подсказка](#)
- Как можно реализовать функцию обмена (swap) в C++? [Подсказка](#)
- Опишите процедуру поиска минимального (максимального) элемента массива. [Подсказка](#)



## Вопросы для самоконтроля

- Что такое рекурсивная функция? [▶ Подсказка](#)
- Какие существуют разновидности рекурсии? [▶ Подсказка](#)
- Когда разумно применять рекурсию? [▶ Подсказка](#)
- Какие задачи традиционно относят к рекурсивным? [▶ Подсказка](#)
- В чём мощь рекурсивного определения? [▶ Подсказка](#)
- Что такое терминальное условие? [▶ Подсказка](#)
- Опишите алгоритм построения рекурсивной функции? [▶ Подсказка](#)
- Как рекурсивно реализовать вычисление факториала? Чисел Фибоначчи? [▶ Подсказка](#)
- В чём суть головоломки "Ханойские башни"? [▶ Подсказка](#)
- Как реализуется программа для "Ханойских башен"? [▶ Подсказка](#)

## Вопросы для самоконтроля

## 1 Простейшие структуры данных

- Введение
- Массивы
- Записи

## 2 Простейшие алгоритмы

- Введение
- Обмен

- Минимальный и максимальный элементы
- Инвертирование
- Группировка

## 3 Автоматный подход

## 4 Рекурсивные задачи

## 5 Вопросы для самоконтроля

## 6 Глоссарий

## 7 Библиографический список

# Глоссарий I

**Массив** множество однотипных элементов, расположенных в смежных ячейках памяти. 5

**Примитивный способ организации** способ, требующих простых (и прямых) операций доступа к элементам. 8

**Структура(запись)** множество разнотипных элементов, расположенных в смежных ячейках памяти. 6

## 1 Простейшие структуры данных

- Введение
- Массивы
- Записи

## 2 Простейшие алгоритмы

- Введение
- Обмен

- Минимальный и максимальный элементы
- Инвертирование
- Группировка

## 3 Автоматный подход

## 4 Рекурсивные задачи

## 5 Вопросы для самоконтроля

## 6 Глоссарий

## 7 Библиографический список

# Библиографический список I



Кормен Т., Лейзерсон Ч., Ривест Р.

*Алгоритмы: построение и анализ*

МЦНМО, Москва, 2000



Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ.

2-е изд. — М.: «Вильямс», 2006



Википедия

<http://ru.wikipedia.org/wiki/Алгоритм>



Википедия

[http://ru.wikipedia.org/wiki/Список\\_алгоритмов](http://ru.wikipedia.org/wiki/Список_алгоритмов)



*Серджвик Р.*

Фундаментальные алгоритмы на C++. Части 1-4

*Diasoft, 2001*

## Библиографический список II



*Седжвик Р.*

Фундаментальные алгоритмы на С. Анализ/Структуры  
данных/Сортировка/Поиск  
*СПб.: ДиаСофтЮП, 2003*



*Седжвик Р.*

Фундаментальные алгоритмы на С. Алгоритмы на графах  
*СПб.: ДиаСофтЮП, 2003*



*Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.*  
Издательский дом «Вильямс», 2000



*Кнут Д.*

Искусство программирования, том 1. Основные алгоритмы  
*3-е изд. — М.: «Вильямс», 2006*



*Кнут Д.*

Искусство программирования, том 2. Получисленные методы  
*3-е изд. — М.: «Вильямс», 2007*

## Библиографический список III



*Кнут Д.*

Искусство программирования, том 3. Сортировка и поиск

*2-е изд. — М.: «Вильямс», 2007*



*Кнут Д.*

Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений

*М.: «Вильямс», 2007*



*Кнут Д.*

Искусство программирования, том 4, выпуск 4. Генерация всех деревьев.

История комбинаторной генерации

*М.: «Вильямс», 2007*



*Керниган Б., Ритчи Д.*

Язык программирования Си

*М.: Финансы и статистика,*



*Подбельский В.В., Фомин С.С.*

Программирование на языке С.

*М.: Финансы и статистика, 2003*

## Библиографический список IV



*Подбельский В.В.*

Практикум по программированию на языке Си.

*М.: Финансы и статистика, 2004*



*Шилдт Г.*

Полный справочник по С.

*М.: Вильямс, 2002.*



*Дейтел Х., Дейтел П.*

Как программировать на С.