



# Анализ программ

Содержание:

1. Зачем читать программы?
2. Виды анализа: «исторический» и логический.
3. Доказательства результативности кода: инварианты, метод математической индукции (ММИ)
4. Паттерны кодирования
5. Техника и примеры анализа

## Зачем читать программы ?

Деятельности, связанные с программированием:

- **обратный инжиниринг** - восстановление «исходника» из двоичного кода
- **реинжиниринг** – «сломать работающий и сделать заново» красиво
- **инспекция** - анализ написанного кода без выполнения с целью логического анализа и понимания

Ситуации, когда это необходимо:

- сопровождение чужого кода
- отладка (программа «должна» работать)
- восстановление контекста после долгого перерыва в работе с кодом
- инверсия процесса проектирования



## Виды анализа

**Исторический (временной)** – наблюдение за исполнением программы (трасса), пошаговая трассировка результата, очевидность, интуиция, обнаружение закономерностей, догадка

**Логический анализ** – выделение фрагментов с известным результатом (паттернов), логический анализ взаимодействия паттернов

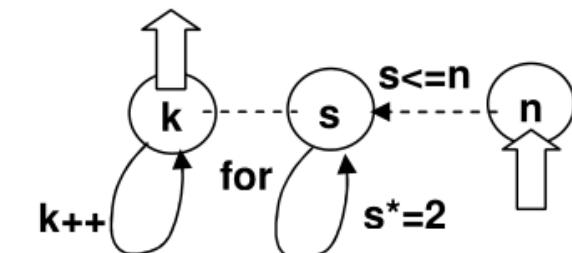
```
int A[10]={3,7,2,4,9,11,4,3,6,3};  
int k,j,s;  
for (i=0,s=A[0]; i<10; i++)  
    if (A[i]>s) s=A[i];
```

```
for (k=0, m=1; m <= n; k++, m = m * 2);
```

I	A[i]	s до if	s после if	Сравнение
0	3	3	3	Ложь
1	7	3	7	Истина
2	2	7	7	Ложь
3	4	7	7	Ложь
4	9	7	9	Истина
5	11	9	11	Истина
6	4	11	11	Ложь
7	3	11	11	Ложь
8	6	11	11	Ложь
9	3	11	11	Ложь
10	Выход		11	

$$n = 100$$

$$k = \text{int}(\log_2 n) + 1$$





# Доказательство результативности кода

Зачем нужно доказательство:

Виды :

- Образное, ОЧЕвидное
- Житейская логика здравого смысла (на базе формальной логики)
- Формальное доказательство: инварианты, метод математической индукции (ММИ)

**Индуктивный подход:** определение общих свойств на основании:

- изучение частных случаев
- интуиция, догадка – формулировка общего свойства
- доказательство общего свойства

**Программа:** общий случай, описание реализации бесконечного множества исполнений на разных наборах данных

**Индуктивный подход к анализу:**

- исторический анализ на конкретном наборе данных
- интуиция, догадка
- доказательство свойств результата



# Доказательство результативности кода

**Инвариант:** общее свойство данных или состояния программы, сохраняемое на всех шагах цикла или рекурсии

**ММИ:** доказательство сохранения общего свойства (инварианта) на последовательности шагов. Утверждение верно всегда, если:

- оно верно при  $i=0$
- если при условии, что оно будет верно на произвольном шаге  $k$ , следует, что оно будет верно на следующем шаге  $k+1$

Пример:

```
for (int i=1, max=A[0]; i<n; i++)  
    if (A[i] > max)  
        max = A[i];
```

если  $\text{max}$  – максимум для  $k=0\dots i-1$

тогда  $\text{max}$  – максимум для  $k=0\dots i$

- инвариант:  $\text{max}$  – максимальное значение
- на первом шаге  $\text{max}=A[0]$  верно для одного элемента массива
- если на  $i$ -ом шаге  $\text{max}$  – максимальное для  $0\dots i-1$ , т.е. на предыдущем шаге, то тело цикла сохраняет этот инвариант:



# Паттерны кодирования

CPROG – 2.4 стандартные программные контексты (старое название)

Паттерн (шаблон) – распространенное решение на уровне идеи, структуры, реализации, которое можно использовать путем подстановки с формальной заменой фрагментов (имен, операций), т.е. с точностью до изоморфизма

Паттерны кодирования - шаблоны на уровне фрагмента отдельной функции (низший уровень кода)

Метафора – образная аналогия. Паттерны кодирования базируются на метафорах исполнения кода.

Доказательство правильности паттерна:

- метафора (очевидность)
- доказательство на уровне неформальной логики (здравый смысл)
- формальное доказательство (инварианты, метод математической индукции)

Паттерны зависят от «набора инструментов»: операций, представления данных:

- массивы: вставка, удаление - сдвиг
- динамическая память и указатели: клонирование, разделение



# Паттерны кодирования

Виды паттернов:

1. «Смысл» переменных в контексте их использования
2. Паттерны циклов
3. Свойства всеобщности и существования
4. Смысл логических операций и выражений
5. Паттерны в различных предметных областях программирования



# «Смысл» переменных

«Смысл» переменной определяется структурой фрагмента кода, где он используется (окружением, контекстом использования), который определяет ее **содержательный (смысловой) результат**. Контекст = сигнатура, по которой фрагмент «виден»

## 1. Присваивание.

1.1 Присваивание – запоминание текущего состояния (свойств) программы в данный момент исполнения

**Великая операция присваивания:** присваивание переводит поведение программы с *инстинктивного* (конечный автомат, «алгоритм без данных», прямой отклик на событие в текущем состоянии) до *рефлективного* (запоминание собственной истории работы, обучение)

Расположение присваивания определяет условия, при которых программа сюда попадает. Запоминание места *последней* перестановки (первое запоминание - защелка)

```
for (i=0; i<n-1; i++)
if (A[i]>A[i+1])           // условие перестановки
{
    c=A[i]; A[i]=A[i+1]; A[i+1]=c;
    b1=i;                  // Запоминание индекса в момент перестановки
}
```



# «Смысл» переменных

## 1.2 Обмен - правило трех стаканов

```
// Обмен значений переменных a,b с использованием переменной c
int a=5,b=6,c;
c=a;           // перелить содержимое первого стакана в пустой (третий) стакан
a=b;           // перелить второй в первый
b=c;           // перелить третий во второй
```

## Вариант со «смешиванием содержимого»

```
// Обмен значений переменных a,b с использованием сложения и вычитания
int a=5,b=6;
a=a+b;         // в переменной a сумма начальных значений a и b
b=a-b;         // в переменной b - (a+b)-b = a
a=a-b;         // в переменной a - (a+b)-a = b
```

## 1.3 Копирование, перенос.

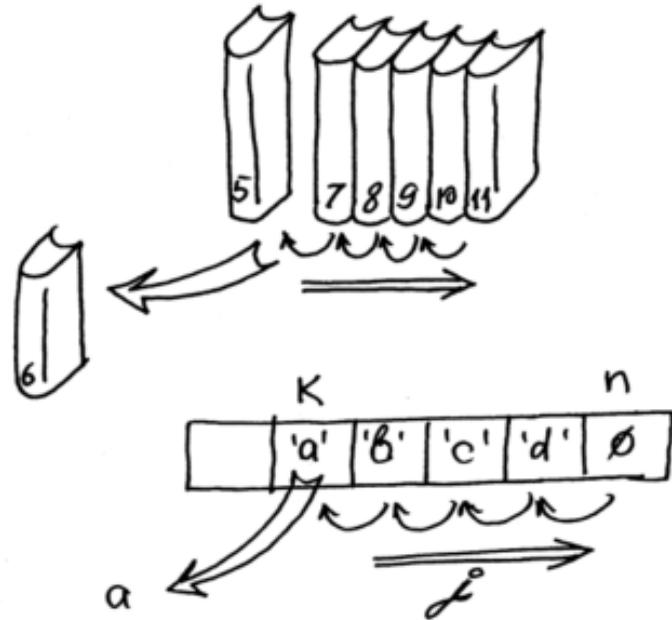
```
for(j=0;n!=0;j++){
    for (k=0,i=1; i<n; i++)
        if (A[i]<A[k]) k=i;
    B[j]=A[k];
    for (;k<n-1;k++) A[k]=A[k+1];
    n--;
}
```



# «Смысл» переменных

## 1.4 Сдвиг, вставка, удаление («плотный» массив) $A[i]=A[i+1]$

```
// Удаление k-го элемента последовательности
int A[]={1,7,3,4,7,6,3,7,4,3}, n=10;
int k=2;
int vv=A[k];
for (int i=k;i<n-1;i++) A[i]=A[i+1];
n--;
//---Вставка k-го элемента в последовательность
int A[20]={1,7,3,4,7,6,3,7,4,3}, n=10;
int k=2,vv=15;
for (int i=n-1;i>=k;i--) A[i+1]=A[i];
A[k]=vv;
n++;
```



Варианты:

- неплотный массив «с дырками». Медицинский факт: данные из файла не удаляются, можно только переписать из одного в другой необходимые элементы. Решение: индексная таблица ссылок (адресов) записей в файле
- последовательности элементов, связанные указателями (адресами) – списки.



# «Смысл» переменных

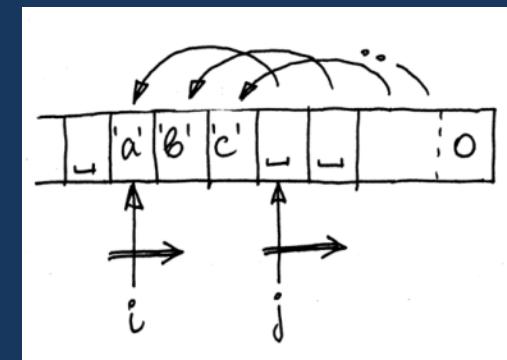
## Вариации вставки/удаления

- вставка «жонглированием»

```
//-----
int A[20]={1,7,3,4,7,6,3,7,4,3}, n=10;
int k=2,vv=15,vv1;
for (int i=k;i<=n;i++){
    vv=A[i];           // Текущий «подбросить вверх»
    A[i]=vv;           // На его место поместить новый
    vv=vv1; }          // «Подброшенный» будет вставляться на следующую
n++;                  // позицию
```

- удаление группы элементов параллельным переносом (паттерн – независимые индексы)

```
//----- Удаление слова с заданным номером
void CutWord(char c[], int n){
int j=0;                      // j - индекс конца слова
for (j=0; c[j]!=0; j++)
    if (c[j]!=' ' && (c[j+1]==' ' || c[j+1]==0))
        if (n==0) break;      // Обнаружен конец n-го слова
    if (n==-1 && c[j]==0){  // Действительно был выход по концу слова
        for (int i=j; i>=0 && c[i]!=' '; i--)
            i++;
        for(j++; c[j]!=0; i++, j++)
            c[i]=c[j];
        c[i]=0;
    }
}}
```





# «Смысл» переменных

## 1.5 Индекс (место) элемента с определенным свойством



# «Смысл» переменных

## 2. Переменная – счетчик (количество)

- `cnt = 0` – сброс счетчика (когда ?)
- `if (...) cnt++;`

```
for (i=0; i<n; i++){  
    for (j=k1=k2=0; j<n; j++)  
        if (c[i] != c[j])  
            { if (c[i] < c[j]) k1++; else k2++; }  
    if (k1 == k2) return i;  
}  
return -1;
```

- $k_1, k_2$  – количество чего ?
- если равны – выход.
- паттерн вложенных циклов, во внутреннем цикле внешний индекс **неподвижен**
- внутренний цикл сравнивает все  $c[j]$  с единственным  $c[i]$ , т.е. определяет свойство для  $c[i]$
- вывод: количество элементов, меньших ( $k_1$ ) и больших ( $k_2$ ) для  $c[i]$
- $c[i]$  – **истинная медиана**, количество меньших и больших одинаково
- пример: 7 3 5 2 9 **4** 1



# «Смысл» переменных

Переменная – счетчик. Сортировка подсчетом (распределяющая): количество элементов, меньших данного – место (индекс) элемента в выходной последовательности

```
void sort(int in[],int n){  
    int i,j,cnt;  
    int *out=new int[n];  
    for (i=0; i<n; i++){  
        for ( cnt=0, j=0; j<n; j++)  
            if (in[j] < in[i]) cnt++;  
        else  
            if (in[j]==in[i] && j<i) cnt++;  
        out[cnt]=in[i];  
    }  
    for (i=0; i<n; i++) in[i]=out[i];  
    delete []out;  
}
```

счетчик => место



# «Смысл» переменных

## 3. Переменная – накопитель суммы (произведения)

- `sum = 0;`
- `for() {... sum = sum + k; }`

```
for (n=a, s=0; n!=0; n=n/10)
    { k=n%10; s=s+k; }
```

Сумма цифр числа

```
for (s=0,i=0; i<n; i++){
    if (A[i]<0) continue;
    if (A[i]==0) break;
    s=s+A[i];
}
```

Сумма положительных до первого 0

```
for (s1=0,s2=0,i=0,j=n-1; i<=j;){
    if (s1<s2) s1+=A[i],i++;
    else s2+=A[j],j--;
}
return i;
```

«Центр тяжести» массива, сумма слева и справа, добавление к меньшему

```
for (s=0,i=0; i<n; i++){
    if (i%2==0)      s=s+A[i];
    else s=s-A[i];
}
```

Разность сумм на четных и нечетных позициях



# «Смысл» переменных

## 3. Переменная – минимум / максимум

- начальное значение – меньше меньшего или больше большего
- если минимум не из всех, то возможна ситуация отсутствия (минимум из положительных при всех отрицательных) => защелка
- if ( $s > max$ )  $max = s$
- можно запоминать не значение, а его местоположение (индекс)

```
// Поиск кратчайшего пути - алгоритм Дейкстры
void min_way(int N, int **M){
    int i,k,j;
    int *P=new int[N],*D=new int[N];
    for (i=0;i<N;i++) P[i]=0,D[i]=1000000;      // Облако пустое
    D[0]=0;                                         // Расстояния – бесконечность, начальная - 0
    while(1){
        for (k=-1,i=0;i<N;i++){
            if (P[i]==1) continue;                // В облаке
            if (k== -1 || D[i] < D[k])           // Ближайшая вне облака
                k=i; }                           // Нет вершин – все в облаке
            if (k== -1) break;                   // Включить в облако
            P[k]=1;
            for (i=0;i<N;i++){
                if (M[k][i]==0) continue; // Коррекция путей к соседям
                if (D[k] + M[k][i] < D[i])   // - если расстояние уменьшается
                    D[i] = D[k] + M[k][i];
            }
    }
}
```



# «Смысл» переменных

## 4. Переменная – признак

- логическая переменная – наступление **события**, обнаружение **свойства**
- устанавливается, сбрасывается, проверяется

```
for (i=0,a=2; a<v; a++){  
    for (s=0,n=2; n<a; n++)  
        if (a%n==0) { s=1; break; }  
    if (s==0) A[i++]=a;  
}  
A[i]=0;
```

- $s_1$  – признак, а делится на  $n$
- внешний цикл перебирает  $a$  от 2 до  $v$
- внутренний цикл устанавливает признак в 0 и перебирает  $n$  от 2 до  $a-1$
- $s_1=1$  число  $a$  делится на число в диапазоне от 2 до  $a-1$  делится (вывод – имеет делитель)
- после внутреннего цикла  $s==0$  – нет делителей, простое



# «Смысл» переменных

## 5. Переменная – защелка

- срабатывает 1 раз за цикл
- защелкой может быть значение переменной, интерпретируемое как признак (индекс  $k < n$  – элемент определен,  $k = n$  – не найден)

```
for (k=0; k<n && A[k]<0;k++)           // Пропуск до первого положительного
for (i=k; i<n; i++){                     // Цикл среди оставшихся
    if (A[i]<0) continue;                 // Пропуск отрицательных
    if (A[i]<A[k]) k=i;                  // Сравнение «в пользу минимального»
}
```

- индекс минимального из положительных
- для сравнения нужно **два** элемента
- $k = -1$  – значение – защелка, не было положительных

```
for (i=0,k=-1; i<10; i++) {           // k=-1 – защелка, не обнаружен положительный
    if (A[i]<0) continue;             // пропуск отрицательных
    if (k== -1) k=i;                  // срабатывает защелка на первом положительном
    else if (A[i]<A[k]) k=i;        // на втором и последующих - сравнение
```



# Паттерны циклов

## 1. Предыдущий, текущий:

- $A[i-1] A[i] A[i+1]$  - предыдущий, текущий, следующий, соседний
- итерационный цикл a-текущее, b-предыдущее,  $b=a$ ;  $a=\text{текущее}$

```
void sort(int A[], int n){  
    int i,found;  
    do {  
        found =0;  
        for (i=0; i<n-1; i++)  
            if (A[i] > A[i+1]) {  
                int cc = A[i]; A[i]=A[i+1]; A[i+1]=cc;  
                found++;  
            }  
    } while(found !=0);  
    // Поиск чисел с возрастающими цифрами  
    void main(){  
        for (int a=100000; a<200000; a++){  
            int n,k,kp=10,m=1; // m - признак убывания  
            for(n=a;n!=0;n=n/10){ // kp - предыдущая цифра  
                k=n%10; // k - очередная цифра  
                if (k>=kp){ m=0; break; } // нет убывания - выход  
                kp=k; // текущая стала предыдущей  
            }  
            if (m!=0) printf("%d\n",a);  
        }  
    }
```



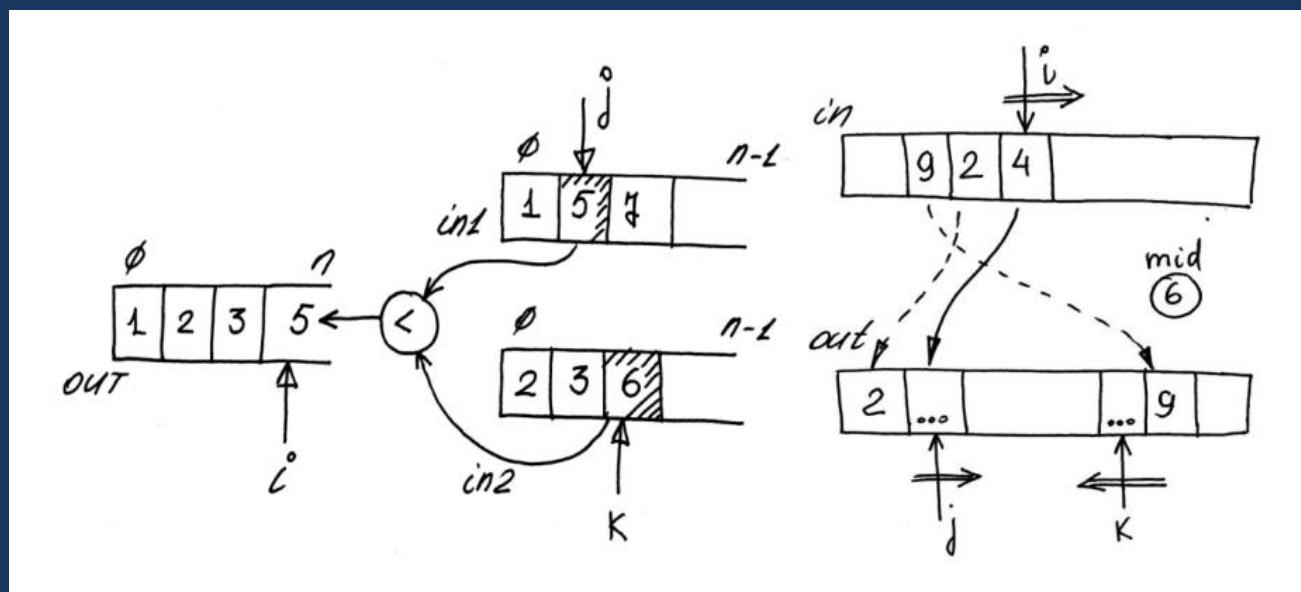
# Паттерны циклов

2. Индекс – независимый движок, «степень свободы»

- равномерное (в заголовке цикла), неравномерное (условие) движение
- общие «балансы» перемещений
- лучше формально независимые индексы, чем сложные вычисления зависимостей

Слияние – соединение двух упорядоченных последовательностей в одну за один цикл

Разделение – «разбрасывание» элементов последовательности в две в зависимости от  $>$  или  $\leq$  медианы (границного значения)





## Паттерны циклов

## Сортировка рекурсивным слиянием:

- массив делится на две равные части
  - рекурсивно сортируются
  - слияние во временный массив и заливка обратно:
    - первая закончилась – из второй
    - вторая закончилась – из первой
    - иначе минимальную из пары текущих



# Паттерны циклов

## 3. Результат цикла – место его остановки.

Минимальный из положительных – цикл до первого положительного

```
for (k=-1,i=0; i<n && A[i]<0;i++){           // Пропуск до первого положительного
    for (; i<n; i++){
        if (A[i]<0) continue;                  // Цикл среди оставшихся
        if (A[i]<A[k]) k=i;                   // Пропуск отрицательных
    }                                         // Сравнение «в пользу минимального»
```

Пословная обработка: цикл по пробелам и цикл по символам слова

```
----- Поиск слова максимальной длины - пословная обработка
// Структурная логика – 3 цикла: просмотр слов, пробелов и символов
int find(char in[]){
    int i=0, k, m, b;
    b=-1; m=0;
    while (in[i]!=0) {                      // Цикл пословного просмотра строки
        while (in[i]==' ') i++;             // Пропуск пробелов перед словом
        for (k=0;in[i]!=' ' && in[i]!=0; i++,k++); // Подсчет длины слова
        if (k>m){                         // Контекст выбора максимума
            m=k; b=i-k; }                  // Одновременно запоминается
    }
    return b; }
```



# Паттерны циклов

4. Вложенные циклы – принцип относительности:

- во внутреннем цикле индексы внешнего **неподвижны**
- сложение индексов внешнего (*i*) и внутреннего (*j*) циклов *i+j* означает просмотр во внутреннем цикле фрагмента, начинающегося с *i*-ой позиции

Пример: поиск одинаковых фрагментов в строке:

- двойной цикл – для каждой пары *i,j*
- попарное сравнение символов в фрагментах, начинающихся с *i*-го и *j*-го символов

```
int F12(char c[]){
for (int i=0; c[i]!=0; i++){
    if (c[i]==' ') continue;
    for (int j=i+1; c[j]==c[i]; j++);
    for (; c[j]!=0; j++){
        for (int k=0; i+k<j && c[i+k]==c[j+k] k++);
        if (k>=4) return i;
    } } return -1; }
```



# Паттерны циклов

5. Выбор «системы координат» при сложном движении в двойном/тройном цикле:

- двойной цикл
- один цикл с моделированием перехода на следующий шаг внешнего цикла  
`j++; if (j==N) { i++; j=0; }`

Пример: циклическое слияние в одном цикле из двух массивов группами по s элементов. Общее начало групп – k сдвигается при достижении конца обоих групп. Сложение индексов – предыдущий паттерн

```
i1=i2=0;
for (i=0,k=0; i<n; i++){
    if (i1==s && i2==s)
        k+=s,i1=0,i2=0;
    if (i1==s || k+i1==n1) A[i]=B2[k+i2++];
    else
        if (i2==s || k+i2==n2) A[i]=B1[k+i1++];
        else
            if (B1[k+i1 ] < B2[k+i2 ]) A[i]=B1[k+i1++];
            else A[i]=B2[k+i2++];
}
```



# Паттерны циклов. Инвариант цикла

Цикл – «одицетворение» метода математической индукции, индуктивного подхода:

- шаг цикла во время исполнения (RunTime) – конкретный шаг
- шаг цикла программировании (в программе) – абстрактный (любой)

Инвариант цикла: свойство цикла, которое сохраняется при переходе с начала текущего шага к следующему (при условии, соответствующем следующему шагу)

Пример: удаление отрицательных элементов из последовательности: инвариант – индекс указывает на очередной элемент

```
int c[] = { 3,7,-6,-7,5,-8,9 };
int n = sizeof(c) / sizeof(int);
for (int i = 0; i < n;) {
    if (c[i] >= 0)
        i++;
    else{ // Следующее место – то же самое
        for (int j = i; j < n - 1; j++)
            c[j] = c[j + 1];
        n--;
    }
}
for (int i = 0; i < n;i++)
printf("%d ", c[i]);
```

Лайфхак: не красить пол, на котором стоишь: копировать нужные элементы с выходной массив



# Свойства всеобщности и существования

В математике проверяются «параллельно». Используются паттерны:

- счетчик
- признак
- место завершения цикла

```
//----- Проверка свойств «для всех» и «существует» с помощью признака
int i,s=1; // признак «оптимистически» установлен в значение «для всех»
for (i=0;i<n;i++)
    if (A[i]<0) // условие всеобщности нарушается
        {s=0; break; } //бросить признак и выйти
if (s==1) «все >0»
else «существуют <=0»
```

```
//----- Проверка свойств «для всех» и «существует» по месту остановки цикла
for (i=0; i<n && A[i]>0;i++); // «пустой» цикл с выходом по двум условиям
if (i==n) «все >0» // был выход по окончании последовательности
else    «существуют <=0» // был выход по обнаружению неположительного
```



# Метафоры логических операций

- AND – одновременно оба...
- OR – хотя бы один
- XOR – несовпадение
- $! (a \&\& b) \Rightarrow !a \parallel !b$
- $! (a \parallel b) \Rightarrow !a \&\& !b$

```
// Цикл завершается при обнаружении пары "меньше 0 - больше 0"
for (i=1; i<20 && !(A[i-1]<0 && A[i]>0); i++);
```

## Паттерны предметных областей

Выражение	Интерпретация
$n \% 10$	младшая цифра числа $n$
$n=n/10$	отбросить младшую цифру $n$
$for (i=0; n!=0; i++, n/=10)$ { ... $n \% 10...$ }	получить цифры числа в обратном порядке
$s = s * 10 + k$	добавить цифру $k$ к значению числа $s$ справа

Поиск – полный перебор, «тупой» алгоритм надежнее сложного:

- линейный перебор
- перебор пар
- комбинаторный перебор (рекурсия)



# Структура данных

Структура данных (СД) – фундаментальное понятие. Примеры:

- это форма хранения и представления информации (*расплывчата*)
- программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных (Википедия) (*т.е. единица алгоритма*)
- это множество элементов данных и связей между ними (*любое*).

**Структура данных** - совокупность взаимосвязанных переменных и их значений:

- переменная – компонента языка, СД – компонента программы
- представление внешней сущности или абстракции (например, группа студентов, стек)
- соглашение о взаимосвязи значений переменных (целостность), соблюдаются всеми функциями, которые работают со СД. Следствия багов (ошибок) – нарушение целостности СД (соглашений)



# Структура данных - последовательность

Пронумерованная (проиндексированная) последовательность однотипных значений. Операции: добавление, вставка по индексу, удаление по индексу, получение по индексу.

Реализация в «плотном» конечном массиве:

- массив + переменная-счетчик элементов
- массив со значением-ограничителем (строка)

```
// Последовательность со счетчиком
#define SZ 1000
int A[SZ];
int n=0;

void add(int vv){ if (n!=SZ-1) A[n++]=vv; }
int get(int k){ return k>=n ? 0 : A[k]; }
int remove(int k){
    if (k>=n) return 0;
    int vv=A[k];
    for (int j=k;j<n-1;j++) A[j]=A[j+1];
    n--;
    return vv;
}
int insert(int vv, int k){
    if (n==SZ-1 || k>=n) return 0;
    for (int j=n-1;j<=k;j--) A[j+1]=A[j];
    A[k]=vv;
    n++;
}
```



# Техника анализа

1. Логический анализ: выделение паттернов, смысл переменных, логический вывод результата
2. Исторический анализ, взаимодействие паттернов. Именно *не единиц языка, а фрагментов с известным результатом (смыслом)*. Логический вывод результата
3. Исторический анализ, наблюдение, догадка, инварианты. Доказательство (ММИ). Логический анализ при доказанных инвариантах.

Данные для исторического анализа:

- *минимизация размерности, когда эффект еще наблюдается*
- *не должны быть вырожденными (1,2,3,4,5 – ???)*



## Техника анализа

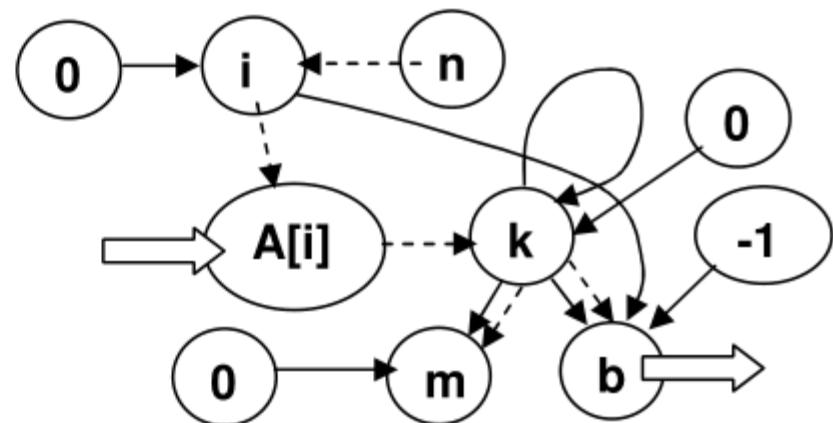
## Определение входа-выхода программы:

- поток данных – граф преобразований значений переменных (присваивания)
  - результат – последнее по времени исполнения присваивание
  - вход – отсутствует в левой части присваивания (начало потока данных)

```

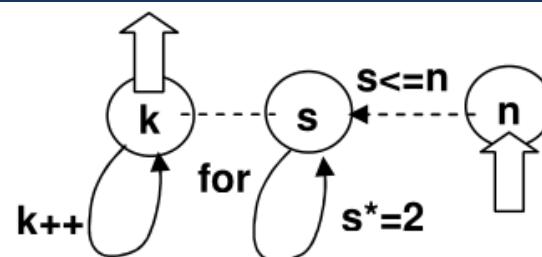
for (i=0,m=0,b=-1; i<n; i++)
    if (A[i]>0) k++;
    else {
        if (k>m) { m=k; b=i-k+1; }
        k=0;
    }

```



## Неявная взаимозависимость через исполнение кода

```
for (k=0, m=1; m <= n; k++, m = m * 2);
```

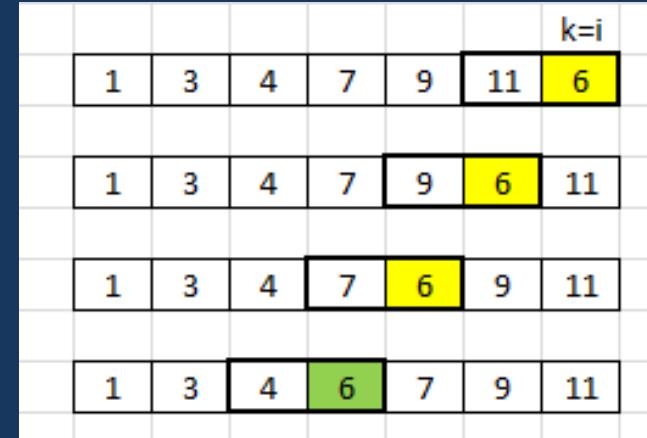




# Примеры анализа

Сложности, когда неизвестен **инвариант** цикла

```
void sort(int in[],int n) {  
    for ( int i=1; i<n; i++)  
        for ( int k=i; k !=0 && in[k] < in[k-1]; k--) {  
            int c=in[k]; in[k]=in[k-1]; in[k-1]=c;  
        }  
}
```



Логический анализ:

- внешний цикл в прямом направлении
- внутренний – в обратном, начиная с текущего во внешнем
- в теле – обмен с предыдущим
- в заголовке - ограничение по границе диапазона и *по сравнению текущего с предыдущим*
- логически результат не доказуем*, тело цикла меняет содержимое массива, которое определяет ограничение цикла

Индуктивный подход: **догадка, наблюдение**: в левой части – **упорядоченная последовательность**:

- очередной меняется с предыдущим, пока не встретит меньшего себя или на достигнет дна – погружение
- свойство упорядоченности ретранслируется на следующий шаг



# Примеры анализа

Сложности, когда неизвестен **инвариант** цикла (еще один пример)

```
for (i=0,a=2; a<v && i<m-1 ; a++){
    for (s=0,j=0; j<i; j++)
        if (a%A[j]==0) { s=1; break; }
        if (s==0) A[i++]=a;
    }
A[i]=0;
```

Логический анализ:

- внешний цикл перебирает  $a$  в диапазоне от 2 до  $v$
- внутренний цикл делит очередное  $a$  на все элементы массива
- признак  $s=1$  – делится нацело на один из...
- после проверки,  $s==0$ , не делится ни на одно, добавляется в массив
- исходный массив *пуст*

Индуктивный подход: **догадка, наблюдение: в массиве – простые числа**

- очередное число является простым, если не делится ни на одно предыдущее простое число (не имеет простых делителей)
- *это же самое и определено при логическом анализе*



# Примеры анализа

Сложности, когда неизвестна **основная идея** (метафора)

```
void sort(int in[], int a, int b){  
    int i,j,mode;  
    if (a>=b) return;  
    for (i=a, j=b, mode=1; i < j; mode >0 ? j-- : i++)  
        if (in[i] > in[j]) {  
            int c = in[i]; in[i] = in[j]; in[j]=c;  
            mode = -mode;  
        }  
    sort(in,a,i-1); sort(in,i+1,b);}
```

Не зная, что это сортировка:

- массив с диапазоном (a,b)
- диапазон = 0 - выход
- индексы навстречу с краев, неравномерно (i,j), пока не встретятся
- рекурсивный вызов для частей диапазона (a,i-1) (i+1,b), i-ый элемент исключается
- обмен текущих крайних элементов, если они не в порядке возрастания
- переменная mode, инвертируется после обмена, в зависимости от значения сдвигается левая или правая граница.



# Примеры анализа

Индукция: догадка и доказательство общей идеи на основе частного «исторического» анализа

a	b	mode				
7	4	9	2	6	1	обмен
6	4	9	2	7	-1	j--
6	4	9	2	7	-1	обмен
2	4	9	6	7	1	i++
2	4	9	6	7	1	i++
2	4	9	6	7	1	обмен
2	4	6	9	7	-1	j--
a,b		Рекурсия				
2	4	9	7			

Догадка: слева от точки встречи находятся значения меньшие, справа – большие (частичная упорядоченность). **Медиана, рекурсивное разделение.**



# Примеры анализа

## Знание (догадка), что это сортировка:

- результатом вызова функции для любого диапазона будет отсортированный диапазон
- индуктивное доказательство:
  1. предположим, что это сортировка
  2. в результате разделения получаем два неупорядоченных диапазона, со свойством <медианы, >медианы ( $in[i]$ )
  3. рекурсивные вызовы их сортируют
  4. весь диапазон, исходя из 2,3 – отсортирован
  5. предположение о сортировке верное

БЕЗ ПРЕДПОЛОЖЕНИЯ НЕТ ДОКАЗАТЕЛЬСТВА

## Быстрая сортировка – идея, которая работает:

- значение справа (под  $j=b$ ) – медиана
- mode – край, где находится медиана, 1 – справа, -1 – слева
- если медиана слева и правое меньше ее – обмен ( $in[i] > in[j]$ )
- если медиана справа и левое меньше ее – обмен ( $in[i] > in[j]$ )



# Примеры анализа

## Более внятное разделение:

- среднее арифметическое – медиана
- два движка, оставляют слева и справа элементы меньшие и большие медианы
- «упираются» в пару элементов, расположенных **наоборот**
- меняют их местами

```
void sort(int in[], int a, int b){  
    int i,j,mode;  
    double sr=0;  
    if (a>=b) return;  
    for (i=a; i<=b; i++) sr+=in[i];  
    sr=sr/(b-a+1);  
    for (i=a, j=b; i <= j;)  
    {  
        if (in[i]< sr) { i++; continue; }  
        if (in[j]>=sr) { j--; continue; }  
        int c = in[i]; in[i] = in[j]; in[j]=c;  
        i++,j--;  
    }  
    if (i==a) return;  
    sort(in,a,j); sort(in,i,b);}
```

## Дефект (ошибка), связанная границной ситуацией:

```
if (i==a) return;
```

- при каких значениях данных это условие срабатывает ?
- к чему приводит его отсутствие ?