Проектирование программ

«Затем я начал разрабатывать транслятор, используя полученные мной знания о технологии программирования, т.е. начал рисовать блок-схему программы, развернув рулон миллиметровой бумаги (миллиметровки) на столе. Когда блок-схема доросла до другого конца стола, я окончательно в ней запутался, и энтузиазм мой угас.»
Романов Е.Л.. PDP-8. ОС на левой коленке

Уровни проектирования:

- **функция (отдельная задача, алгоритм)**
- набор функций, структура данных - библиотека, класс
- функциональная единица: паттерны проектирования, система взаимодействующих классов
- проект (ограниченный < 10^4 SLOC, средний < 10^5 SLOC, большой)

Вопросы:

- жизненный цикл проектирования программы
- «историческое» программирование
- структурное программирование
- приемы (паттерны) программирования

Проектирование программ
Жизненный цикл проектирования программы

- Постановка задачи – результат с определенным свойством
- Идея решения
- Образная модель
- «Запчасти» - переменные со «смыслом», паттерны (конструкции)
- Выстраивание программы из «запчастей» = собственно технология программирования
- Программа – общий случай, граничные ситуации

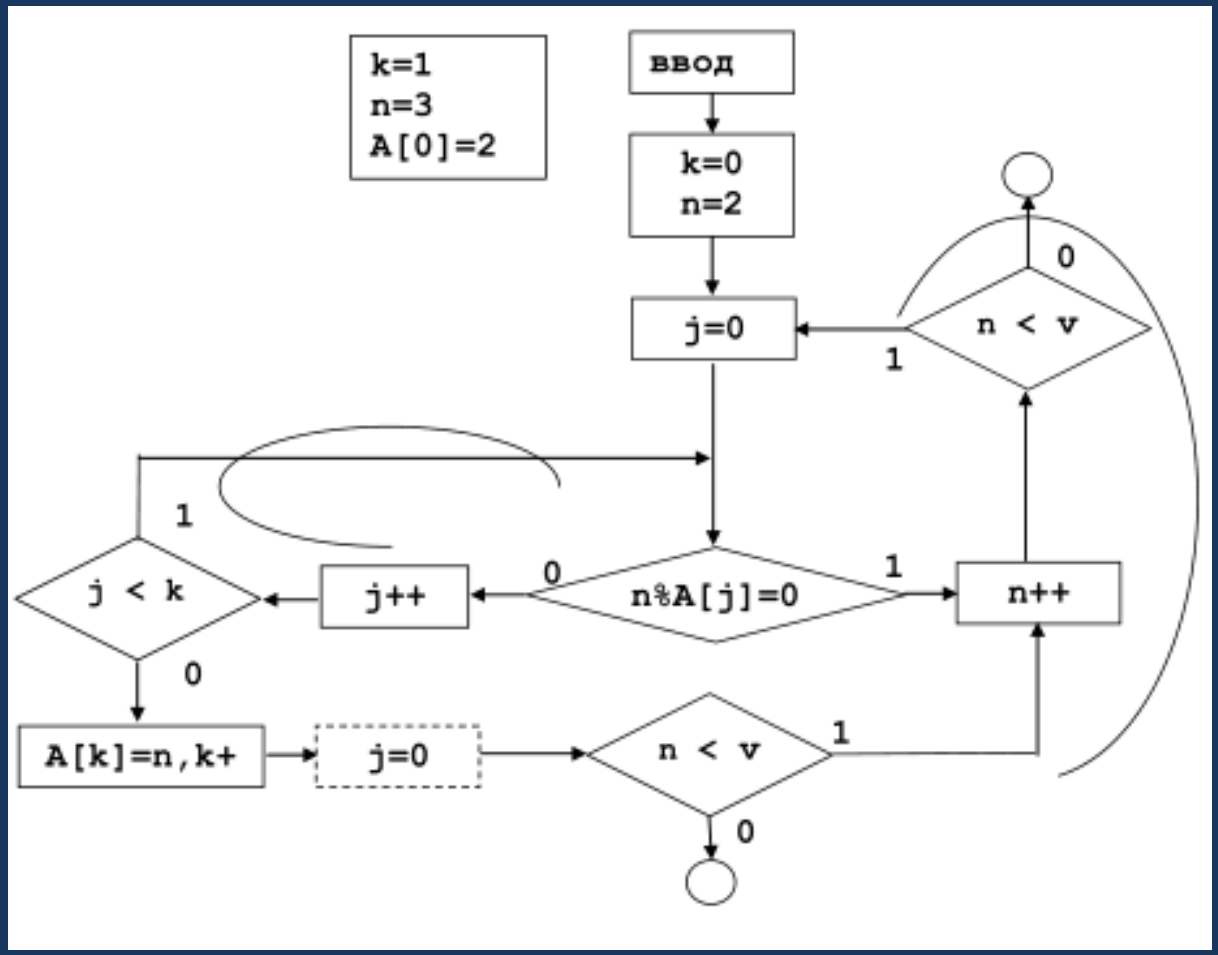
Проектирование программ
«Историческое» программирование

Тезисы:

- аналог проектирования – блок-схема
- «лесом еду – лес пою», решение по текущей ситуации
- последовательность записи кода соответствует последовательности его выполнения
- программирование «шиворот навыворот» - внутренние конструкции оказываются снаружи
- ориентирован на goto (мы это уже делали…)
- при возврате к началу меняется состояние программы
- аналог – диаграмма состояний, автоматная модель
- используется для первичного понимания работы программы
- логика операторов языка программирования – структурированная (1 вход – 1 выход, вложенность) - не ложится напрямую на «историческое» программирование

Проектирование программ
«Историческое» программирование

Пример: поиск простых чисел путем проверки делимости на уже найденные



Проектирование программ
«Историческое» программирование

Шаг 1. Начало. 2 – простое число, Шаг 2 – удалось с goto

```
int main(){
    int A[100]={2};
    int k=1,n=2, v;
    printf("v=");
    scanf("%d",&v);
    for (int i=0;i<k;i++)
        printf("%d ",A[i]);
    puts("");
    getchar();
    return 0;
    }
```

```
стить, Компилировать файл)  ⊠ | cpp1 (Собрать, Выполнен
v=100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53
```

```
    int j=0;                    // Делим на элементы массива
retry:
    if (n % A[j]!=0){           // Не делится
        j++;                    // к следующему элементу
        if (j==k){              // Кончились простые
            A[k++] = n;         // Добавить простое
            j=0; n++;
            if (n==v){          // Конец работы
                for (int i=0;i<k;i++)
                printf("%d ",A[i]);
                puts("");
                return 0;
                }
            }
        goto retry;
        }
    else{                       // Делится
        j=0; n++;               // Взять следующее n
        if (n==v){
            for (int i=0;i<k;i++)
            printf("%d ",A[i]);
            puts("");
            return 0;
            }
        goto retry;
        }
```

Проектирование программ
«Историческое» программирование

Шаг 3. Замена goto на цикл Шаг 4. Ограничиваем цикл

```
int j=0;                    // Делим на элементы массива
while (true){
    if (n % A[j]!=0){       // Не делится
        j++;                // к следующему элементу
        if (j==k){          // Кончились простые
            A[k++] = n;     // Добавить простое
            j=0; n++;
            if (n==v)
                break;
            }
        }
    else{                   // Делится
        j=0; n++;           // Взять следующее n
        if (n==v)
            break;
        }
    }
for (int i=0;i<k;i++)
    printf("%d ",A[i]);
puts("");
```

```
int j=0;
while (n!=v){
    if (n % A[j]!=0){
        j++;
        if (j==k){
            A[k++] = n;
            j=0; n++;
            }
        }
    else{
        j=0; n++;
        }
    }
```

Основной недостаток – вся программа в одном цикле

Проектирование программ
«Историческое» программирование. Как надо

Идея и образная модель – понятны

«Запчасти»

1. Граница работы - v
2. Массив накопленных простых A[], количество накопленных – k
3. Начальное состояние A[0]=2, k=1, n=3
4. Проверяемое число n
5. Проверяются все n в диапазоне 3..v-1 - цикл
6. Свойство всеобщности – n не делится ни на одно A[j] j=0..k-1- цикл
7. Признак для 6 или паттерн «точка останова цикла» j==k
8. Граничная ситуация – k=0, n=2 – будет работать и при пустом массиве

```
for (k=0,n=3; n<v && k < N; n++){          // Внешний цикл перебора n
        for (j=0; j<k &&  n%A[j]!=0; j++);  // Внутренний цикл перебора A[] (без тела)
        if (j==k) A[k++]=n;                 // добавить очередной в A[]
        }}                                  // при выходе по второму условию
```

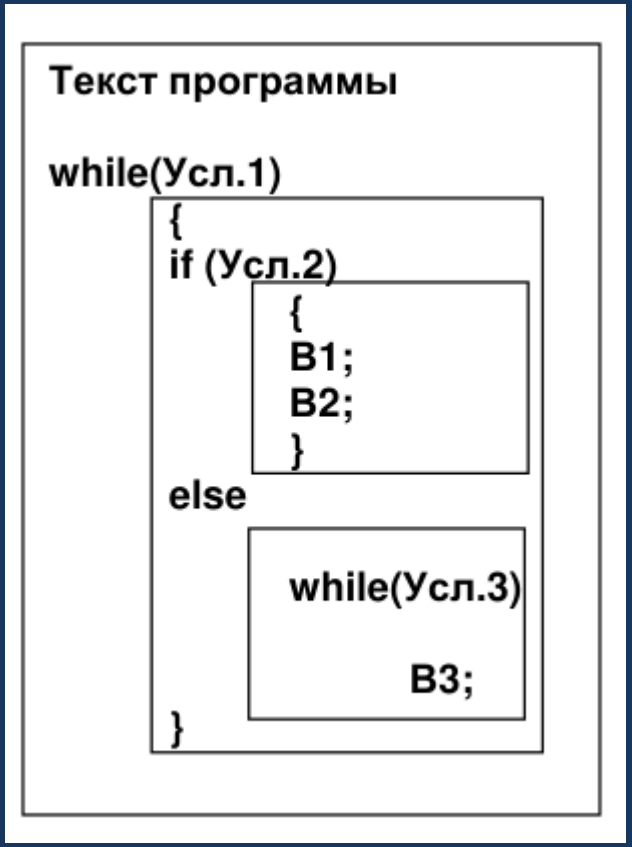
Достоинство: каждый цикл отвечает за свой процесс

Проектирование программ
Структурное программирование

- 70-годы, Паскаль
- программирование без goto
- структурированный код
  - 1 вход – 1 выход
  - вложенность (принцип «матрешки»)
  - операторы: последовательность (блок), ветвление, цикл
  - «слабые» goto – break, continue, return, throw

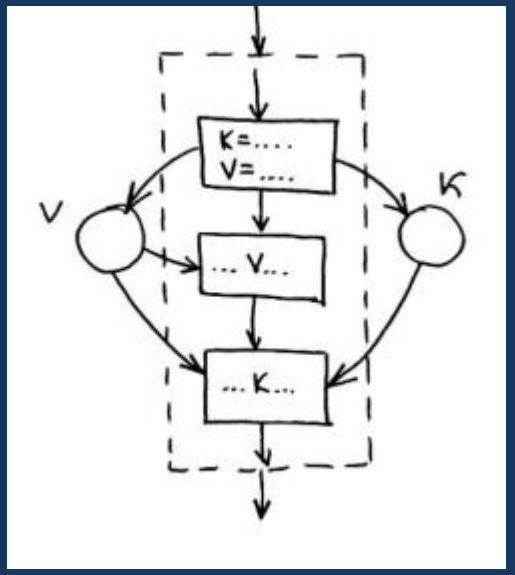
Заповеди структурного программирования

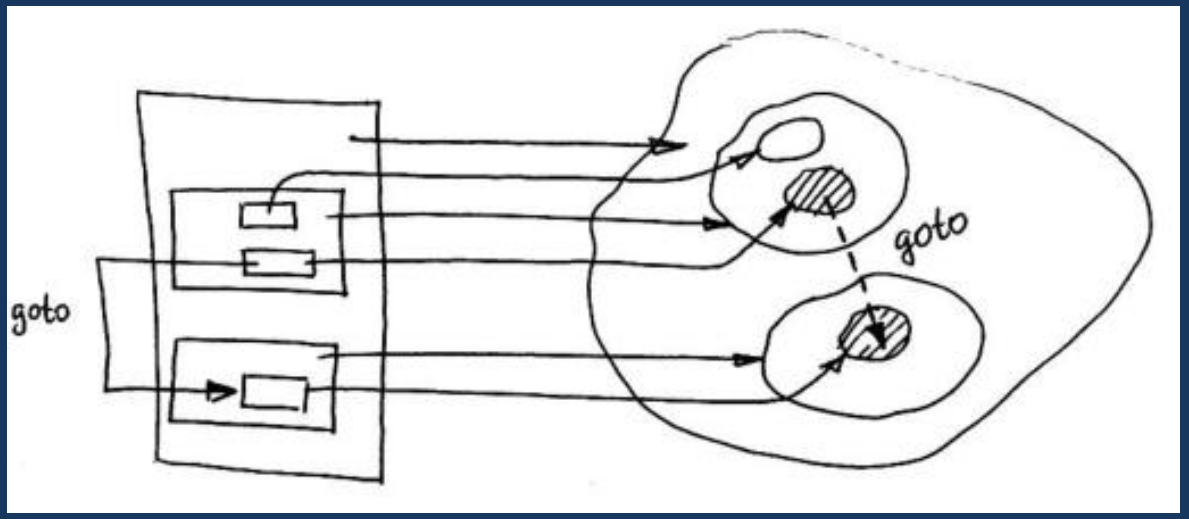
- нисходящее, внешняя конструкция появляется в тексте программы раньше и **не меняется** при включении внутренней
- пошаговое – 1 шаг, замена текстовой формулировки на 1 оператор
- структурное – оператор может быть последовательностью, ветвлением или циклом
- модульное
- одновременное проектирование алгоритма и данных



Проектирование программ
Артефакты структурного программирования

- Конструкция = оператор + текст TODO для тела
- Шаг проектирования: текст TODO = оператор + текст для тела
- Последовательность – действия, связанные результатами, нельзя перепрыгнуть пропасть за 2 шага, линейный алгоритм
- Текст TODO для последовательности = TODO1, TODO2…TODOn
- «среда обитания кода» - условия и инварианты, создаваемые внешними конструкциями





Приемы проектирования

- автоматный подход vs формат, «зашитый» в код программы
- индуктивный подход. Инвариант цикла и рекурсии
- «грязное» программирование vs прототип
- выбор системы координат
- «восходящее» программирование

Приемы проектирования
Автоматный подход vs формат, «зашитый» в код программы

Программа отрабатывает последовательность данных в определенном формате.
Пример: строка слов. Слово – последовательность символов, ограниченная пробелами. Пробелы в начале и конце строки.

Вариант 1. **Переменные состояния,** запоминающие события в программе и другие характеристики просмотренных данных

Вариант 2. **Структура кода отражает правила** следования элементов:
- последовательность – друг за другом
- условие – текущий элемент определяет формат последующих
- цикл – повторение следования элементов

Следствие из **теории алгоритмов**: один и тот же алгоритм (программа) может быть реализована с мин. количеством переменных и мин. количеством операторов.

Приемы проектирования

Автоматный подход и формат, «зашитый» в код программы

Вариант 1. **Переменные состояния,** запоминающие события в программе и другие характеристики просмотренных данных

```
//-----------------------------------------------------44-12.cpp
//---- Поиск слова максимальной длины - посимвольная обработка
// Функция возвращает индекс начала слова или 1, если нет слов
// Логика переменной состояния – n – счетчик символов слова
 int find(char s[]) {
 int i,n,lmax,b;
 for (i=0,n=0,lmax=0,b=-1; s[i]!=0; i++){
          if (s[i]!=' ') n++;            // символ слова увеличить счетчик
          else {                         // перед сбросом счетчика
                    if (n > lmax) { lmax=n; b=i-n; }
                    n=0;                 // фиксация максимального значения
                    }}                   // то же самое для последнего слова
 if (n > lmax) { lmax=n; b=i-n; }
```

Приемы проектирования

Автоматный подход и формат, «зашитый» в код программы

Вариант 2. **Структура кода отражает правила** следования элементов

```
//-----------------------------------------------------44-13.cpp
//---- 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
     }                                   // индекс начала
 return b; }
```

Приемы проектирования

Автоматный подход и формат, «зашитый» в код программы

Вариант 1. **Переменные состояния,** запоминающие события в программе и другие характеристики просмотренных данных (Удаление комментариев. Java)

```
//------------------------------------------------------------------------------------------ 84_TestExamples
  public String F1(String in){                  // Копирование строки - ПЕРЕМЕННАЯ СОСТОЯНИЯ
      int i,j,cm;                               // Вложенные комментарии отслеживаются
      char c[]=in.toCharArray();
      StringBuffer out=new StringBuffer();
          for (i=j=cm=0; i<c.length; i++) {
          if (c[i]=='/' && i!=c.length-1 && c[i+1]=='*') { cm++; i++; continue; }
          if (c[i]=='*' && i!=c.length-1 && c[i+1]=='/') { cm--; i++; continue; }
          if (cm>0) out.append(c[i]);
          }
      return out.toString();
      }
```

Приемы проектирования

Автоматный подход и формат, «зашитый» в код программы

Вариант 2. **Структура кода отражает правила** следования элементов (Удаление комментариев. Java)

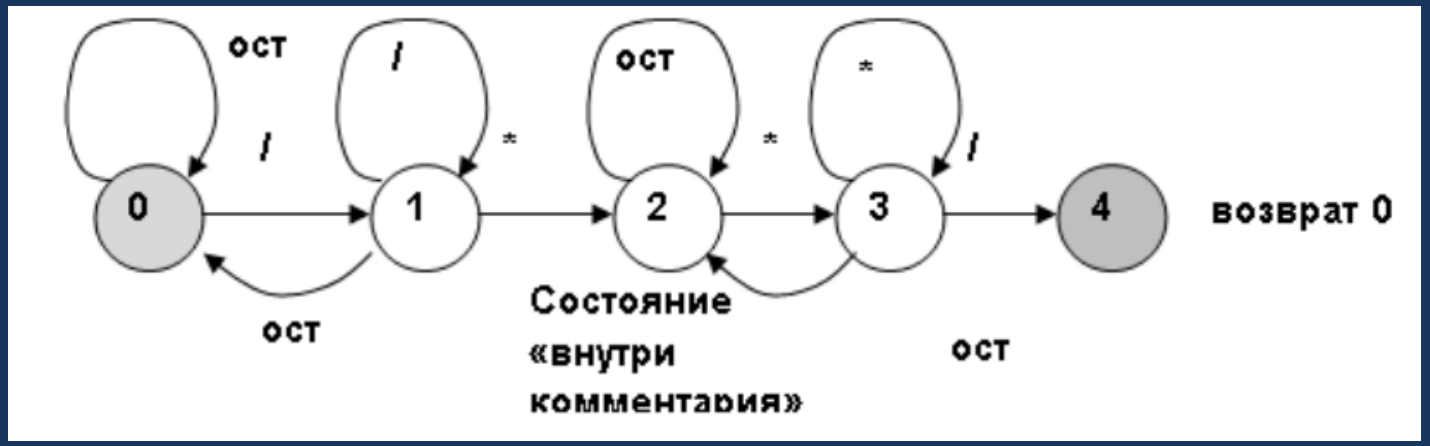
```
//---------------------------------------------------------------------------------------------- 84_TestExamples
    public String F2(String in){          // Распознавание ПО ФОРМАТУ
        int i,j,cm;
        char c[]=in.toCharArray();
        StringBuffer out=new StringBuffer();
            for (i=j=cm=0; i<c.length; i++) {
            if (c[i]=='/' && i!=c.length-1 && c[i+1]=='*') {
                for(i+=2; i<c.length-1 && !(c[i]=='*' && c[i+1]=='/');i++)
                    out.append(c[i]);
                i+=2;
                }
            else i++;
            }
        return out.toString();
        }
```

Приемы проектирования

Автоматный подход и формат, «зашитый» в код программы

Вариант 3. **Формальная система – конечный автомат.** Диаграмма состояний – переходов (табличная реализация). (Удаление комментариев)

Достоинства: поведение автомата записано в таблице (данные), загружается в автоматную модель.



	Ост	/	*
S0	0	1	0
S1	0	0->1	2
S2	2	2	3
S3	2	4	2->3

Приемы проектирования

Автоматный подход и формат, «зашитый» в код программы

Вариант 3. **Формальная система – конечный автомат.** Диаграмма состояний – переходов (табличная реализация). (Удаление комментариев. Java)

```
public int tbl[][]={{0,1,0},{0,1,2},{2,2,3},{2,4,3}};
public String F3(String in){        // Автоматная модель
   int i,beg=0,state=0;
   char c[]=in.toCharArray();
   StringBuffer out=new StringBuffer();
       for (i=state=0; i<c.length; i++) {
      int col=0;
      if (c[i]=='/') col=1;
      if (c[i]=='*') col=2;
      int newState=tbl[state][col];
      //------------- Поведение автомата ------------------------------
      if (state==1 && newState==2) beg=i+1;   // Начало текста комментария
      if (newState==4) {                                  // Конец комментария

       for(int j=beg; j<i-1;j++)
          out.append(c[j]);
       state=0;
       }
    else state=newState;                          // Переход в новое состояние
    }
 return out.toString();
}
```

Приемы проектирования
Индуктивный подход. Инвариант цикла и рекурсии

Индукция: анализ явления от частного к общему:
- наблюдение за частным случаем
- обнаружение закономерностей (опыт, интуиция)
- доказательство закономерности для общего случая

Инвариант: параметрическое свойство, сохраняемое для каждого значения параметра

Инвариант цикла: свойство, сохраняемое в последовательности на каждом шаге цикла, работающего с ней

Ошибки, связанные с инвариантами: ошибки граничных условий
- ошибка первого шага – условия инварианта не соблюдаются перед началом
- «+/- метр от столба» - лишний или недостающий шаг цикла

Приемы проектирования

«Грязное» программирование vs прототип

Прототип: Вариант реализации программы, реализующий структуры, закономерности, алгоритмы, которые будут в дальнейшем соблюдаться.

Рефакторинг: изменение структуры кода без изменения функциональности («качество» кода).

«Тюнинг» программного кода: добавление функционала, не меняющего законов работы прототипа (медицинский принцип «Не навреди»)

«Грязное» программирование: правка кода без соблюдения вышесказанного

Приемы проектирования
Индуктивный подход. Инвариант цикла и рекурсии

Пример. Пословный просмотр строки. Инвариантное утверждение: на каждом следующем шаге находимся на начале следующего слова.

```
//-----------------------------------------37-02.cpp
// Пословная обработка: "1 шаг = 1 слово". "Грязная программа"
void  F1(char in[]){
int i=0,k;
while (in[i]==' ')i++;          // ДЛЯ ПЕРВОГО ШАГА
while(in[i]!=0){                // 1 шаг = 1 слово
        // Начало слова
        while (in[i]!=' ' && in[i]!=0) i++;
        // Конец  слова
        while (in[i]==' ')i++;
        }}
```

Приемы проектирования
Индуктивный подход. Инвариант цикла и рекурсии

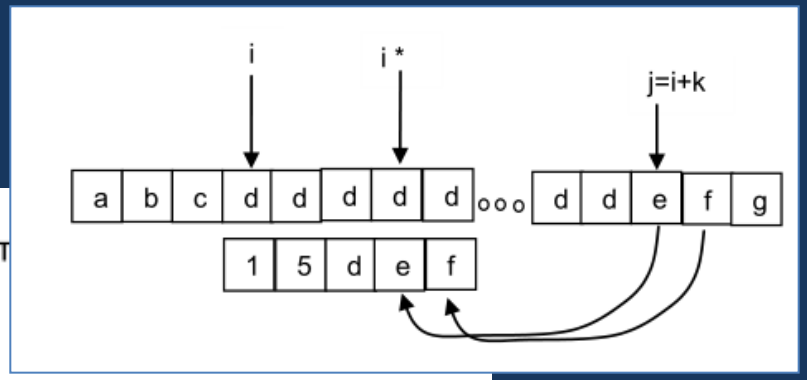
Пример. Замена цепочки повторяющихся символов счетчиком.
Шаг 1. Прототип. Формат, «зашитый в код». Инвариант цикла: 1 шаг – один символ или цепочка повторяющихся

```
//-----------------------------------------37-03.cpp
// Отдельный цикл "измерения" повторений. "Грязная программа"
void  F3(char in[]){
int i,j,k;
for (i=0; in[i]!=0;){
        for(k=1;in[i]==in[i+k];k++);        // Измерить длину цепочки
        if (k>1) printf("%d",k);
        printf("%c",in[i]);
        i=i+k;                              // Шаг цикла – длина цепочки
        }
}
```

в Java – дефект i+k<in.length &&

Приемы проектирования
Индуктивный подход. Инвариант цикла и рекурсии

Пример. Замена цепочки повторяющихся символов счетчиком.
Шаг 2. Сжатие. После сжатия оказываемся на начале «подтянутого» хвоста (инвариант цикла при сжатии строки)
abc|ddddddddefg -> abc7d|efg




```
//-----------------------------------------37-04.cpp
// Отдельный цикл "измерения" повторений. "Сжатие ст
void  F4(char in[]){
int i,j,k;
for (i=0; in[i]!=0;){
        for(k=1;in[i]==in[i+k];k++);          // Измерить длину цепочки
        if (k==1) i++;                        // Нет повторения - пропустить
        else{
                j=i+k;                        // Запомнить начало "хвоста"
                int k1,j1;                    // Записать k в виде символов-цифр
                for (k1=k; k1!=0;k1=k1/10,i++);
                for (j1=i-1; k!=0;k=k/10,j1--)
                        in[j1]=k%10+'0';      // Затереть символы цифрами числа
                i++;                              // «Оставить» один символ
                j1=i;                             // i – начало следующего шага
                do {in[j1++]=in[j]; }             // j1 – куда, j - откуда
                        while(in[j++]!=0);
                }
        }}
```

Приемы проектирования
Индуктивный подход. Инвариант цикла

Вариант 2. Замена цепочки повторяющихся символов счетчиком.
Шаг 2. Заполнение выходной строки: одиночные символы переносятся, повторяющиеся – конвертируются. Инвариант основного цикла: 1 шаг = неповторяющийся символ ИЛИ цепочка повторений

```
// Отдельный цикл "измерения" повторений. Копирование строки
void  F4(char in[], char out[]) {
    int i, j, k, s = 0;       // i,s - независомое перемещение по входной/выходной строке
    for (i = 0; in[i] != 0;) {
        for (k = 1; in[i + k] != 0 && in[i] == in[i + k]; k++); // Измерить длину цепочки
        if (k == 1) out[s++] = in[i++];                         // Нет повторения - копировать
        else {
            char cc = in[i];
            for (int k1 = k; k1 != 0; k1 = k1 / 10, s++);       // Вперед на количество цифр
            for (int j1 = s - 1, k1 = k; k1 != 0; k1 = k1 / 10, j1--)
                out[j1] = k1 % 10 + '0';                        // Цифры в обратном порядке
            out[s++] = cc;
            i += k;           // Пропустить повторение = ИНВАРИАНТ ЦИКЛА
            }
        }
    out[s] = 0;               // Не забыть закрыть выходную строку
    }
```

```
fdc47d10e20fdfgdfdgfdfgd
```

```
char xx[100];
F4((char*)"fdcddddddddddddddddddddddddddddddddddddddddddddddddddddddeeeeeeeeeeffffffffffffffffffffdfgdfdgfdfgd", xx);
puts(xx);
```

“aaaaa” = const char*

Ошибки инварианта:
- ошибка «первого шага»
- +/- «метр от столба»

Приемы проектирования

Выбор системы координат

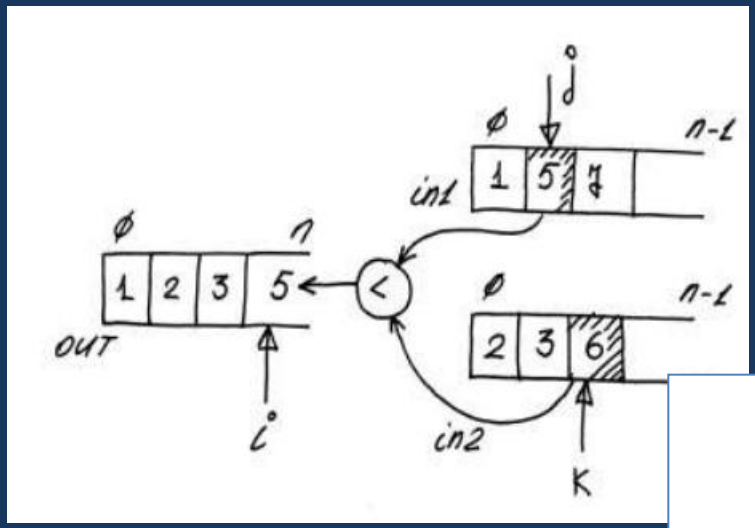
Выбор количества циклов, движков, инвариантов шага

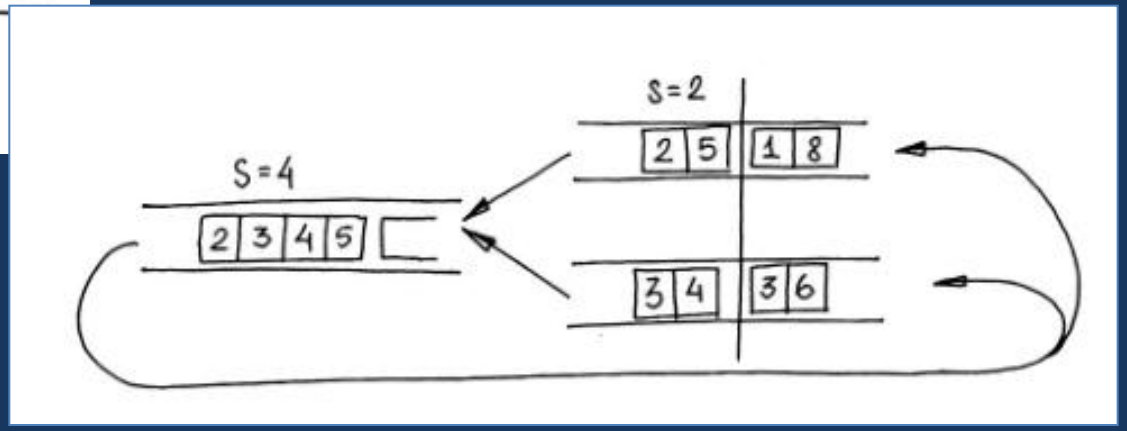
Пример: циклическое слияние

Слияние: соединение двух упорядоченных последовательностей в **одном** цикле дает упорядоченную последовательность двойной длины

Идея: слияние группами по s=1,2,4,8… 2^m элементов

Граничное условие: размерность массива не кратна 2^m





Приемы проектирования

Выбор системы координат

Система координат: линейный цикл слияния двух массивов группами вместо 2 циклов (по группам и внутри групп), движки – k-база, i1,i2-смещения

```
//-------------------------------------------------------------------------46-16.cpp
//------ Циклическое двухпутевое слияние
void sort(int A[], int n){
int i,i1,i2,s,k;
for (s=1; 1; s*=2){                                  // Размер группы кратен степени 2
        int nn=n/s;                                  // Количество групп по s элементов
        if (n%s!=0) nn++;                            // Остаток – есть неполная группа
        int n1=nn/2*s;                               // Деление ближе к середине,
        int n2=n-n1;                                 // но кратно размеру группы
        if (n1<=0 || n2<=0) return;                  // Часть больше целого - выход
        int *B1=new int[n1],*B2=new int[n2];
        for (i=0; i<n1; i++) B1[i]=A[i];             // Разделение на части
        for (i=0; i<n2; i++) B2[i]=A[i+n1];
        i1=i2=0;
        for (i=0,k=0; i<n; i++){                     // Слияние с переходом «скачком»
                if (i1==s && i2==s)                  // при достижении границ обеих
                        k+=s,i1=0,i2=0;              // групп
                if (i1==s || k+i1==n1) A[i]=B2[k+i2++];  // Достигла границы группы или
                else                                 // массива
                if (i2==s || k+i2==n2) A[i]=B1[k+i1++];
                else                                 // Если нет – минимальный из пары
                if (B1[k+i1 ] < B2[k+i2 ]) A[i]=B1[k+i1++];
                else A[i]=B2[k+i2++];
                }
        delete []B1; delete []B2;
        }}
```

Приемы проектирования

Восходящее программирование

- модульное программирование от функции к функции «снизу вверх» от частного к общему
- внутри функции – аналог «исторического» программирования

Пример: сортировка выбором – цикл для первого шага

```
int j,k;
for (k=0,j=1;j<n;j++)                // Поиск минимального -  сохранение индекса
        if (A[j]<A[k]) k=j;
int c=A[0]; A[0]=A[k]; A[k]=c;       // Обмен первого A[0] и минимального
```

Удачная коррекция кода для i-го шага внешнего цикла

```
//-----------------------------------------------------35-03.cpp
//---------- Сортировка выбором – восходящее программирование
// Шаг 2.
 for (i=0;i<n-1;i++){
        int j,k;
        for (k=i, j=i+1;j<n;j++)
                if (A[j]<A[k]) k=j;
        int c=A[i]; A[i]=A[k]; A[k]=c;
        }
```

Пример проектирования

Структурное программирование «из головы»

Задача на вычеркивание: из числа из n цифр вычеркнуть m, чтобы осталось максимальное

- образная модель n=8 m=4 a=54382674
- идея: удалять минимальные – 5**8*67* не работает, ***8*674
- идея: на первом месте должна быть максимальная из возможных, т.е. найти максимальную из n-m+1=5 и вычеркнуть все перед ней ***82674
- идея: после этого задача повторяется для оставшейся части за найденной цифрой с учетом вычеркивания: вычеркнуто 3 – n=4, m=1 a=2674
- представление данных: массив цифр, способ вычеркивания
  - замена на -1 (cprog 3.6)
  - сдвиг при удалении

Образная модель и «запчасти» для второго варианта:

1. на каждом шаге добавляется по 1 цифре в результате сдвига – k – индекс
2. поиск максимума k..k+m+1, относительный индекс s - k…k+s…k+m+1
3. сдвиг на s элементов влево, начиная k+s и оканчивая n-ым справа
4. n-=s+1 m-=s+1
5. цикл 1 повторяется, пока m!=0

Пример проектирования

Структурное программирование «из головы»

1

```
int removeDigits(int a[], int n, int m){
    int rem = n-m;
    return rem;
    }

int main(){
    int A[100]={5,4,3,8,2,6,7,4};
    int ss = removeDigits(A,8,4);
    for (int i=0;i<ss;i++)
        printf("%d ",A[i]);
    puts("");
    getchar();
    return 0;
    }
```

(Собрать)

```
5 4 3 8
```

2

```
int removeDigits(int a[], int n, int m){
    int rem = n-m;
    for(int k=0; m!=0;k++){      // ШАГ 1 - 1,5
        // TODO найти очередную цифру
        }
    return rem;
    }
```

3

```
int removeDigits(int a[], int n, int m){
    int rem = n-m;
    for(int k=0; m!=0;k++){      // ШАГ 1 - 1,5
        // ШАГ 2 - последовательность 2,3,4
        // TODO 2
        // TODO 3
        // TODO 4
        }
    return rem;
```

4

```
int removeDigits(int a[], int n, int m){
    int rem = n-m;
    for(int k=0; m!=0;k++){      // ШАГ 1 - 1,5
        // ШАГ 2 - последовательность 2,3,4
        int s=0;
        // TODO 2
        // TODO 3
        // TODO 4
        n -= s+1;
        m -= s+1;
        }
    return rem;
    }
```

5

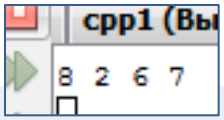
```
int removeDigits(int a[], int n, int m){
    int rem = n-m;
    for(int k=0; m!=0;k++){      // ШАГ 1 - 1,5
        // ШАГ 2 - последовательность 2,3,4
        int s=0;
        for(int j=0;j<m+1;j++)   // 2
            if (a[k+j]>a[k+s])
                s = j;
        // TODO 3
        n -= s+1;                // 4
        m -= s+1;
        }
    return rem;
    }
```

Пример проектирования

Структурное программирование «из головы»

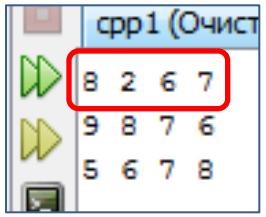
6

```
int removeDigits(int a[], int n, int m){
    int rem = n-m;
    for(int k=0; m!=0;k++){      // ШАГ 1 - 1,5
        // ШАГ 2 - последовательность 2,3,4
        int s=0;
        for(int j=0;j<m+1;j++)       // 2
            if (a[k+j]>a[k+s])
                s = j;
        for (int j=k; j+s<n; j++)    // 3
            a[j] = a[j+s];
        n -= s+1;                    // 4
        m -= s+1;
        }
    return rem;
    }
```



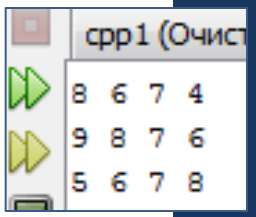
7

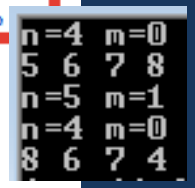
```
int removeDigits(int a[], int n, int m){
    int rem = n-m;
    for(int k=0; m>0;k++){       // ШАГ 1 - 1,5
        // ШАГ 2 - последовательность 2,3,4
        int s=0;
        for(int j=0;j<m+1;j++)       // 2
            if (a[k+j]>a[k+s])
                s = j;
        for (int j=k; j+s<n; j++)    // 3
            a[j] = a[j+s];
        n -= s+1;
        m -= s+1;
        }
    return rem;
    }

int main(){
    int A[100]={5,4,3,8,2,6,7,4};
    int B[100]={9,8,7,6,5,4,3,2};
    int C[100]={1,2,3,4,5,6,7,8};
```



8

```
int removeDigits(int a[], int n, int
    int rem = n-m;
    for(int k=0;m!=0;k++){
        int s=0;
        for(int j=0;j<m+1;j++)
            if (a[k+j]>a[k+s])
                s=j;
        for(int j=k;j+s<n;j++)
            a[j]=a[j+s];
        n -= s; // Сколько осталось ВСЕГО
        m -= s; // Сколько надо удалить
        printf("n=%d m=%d\n",n,m);
        }
    return rem;
    }
```





Ошибка, выяснилась при зависании на C[], была на A[] (+/- метр от столба)

54382674 n=8 m=4 -> s=3

82674 n=4 m=0 выход, надо 8674

12345678 n=8 m=4 -> s=4

5678 n=3 m=-1 – цикл при m!=0