Операторы и логика алгоритма

Общее:

- описание последовательности действий - *алгоритм* в узком смысле
- синтаксические конструкции - *операторы*
- общий архитектурный базис – *система команд процессора*, описание *потока управления*

Особенное:

- присваивание, вызов функций (процедур), ввод-вывод (функции) – синтаксически входят в *выражения*
- *C++* - обработка ошибок (*исключения, throw, try, catch)* – описание альтернативного потока управления, связанного с обработкой сгенерированной в коде ошибки (exception). [cprog 11.3]
- свободные, синтаксически слабо ограниченные конструкции операторов – *подводные камни* - возможные ошибки из-за замены одной допустимой конструкции на другую

Кроме явно определяемых операторов:

Выражение**;** -> Простой оператор

; -> Пустой оператор

Универсальные группы (триады) операторов

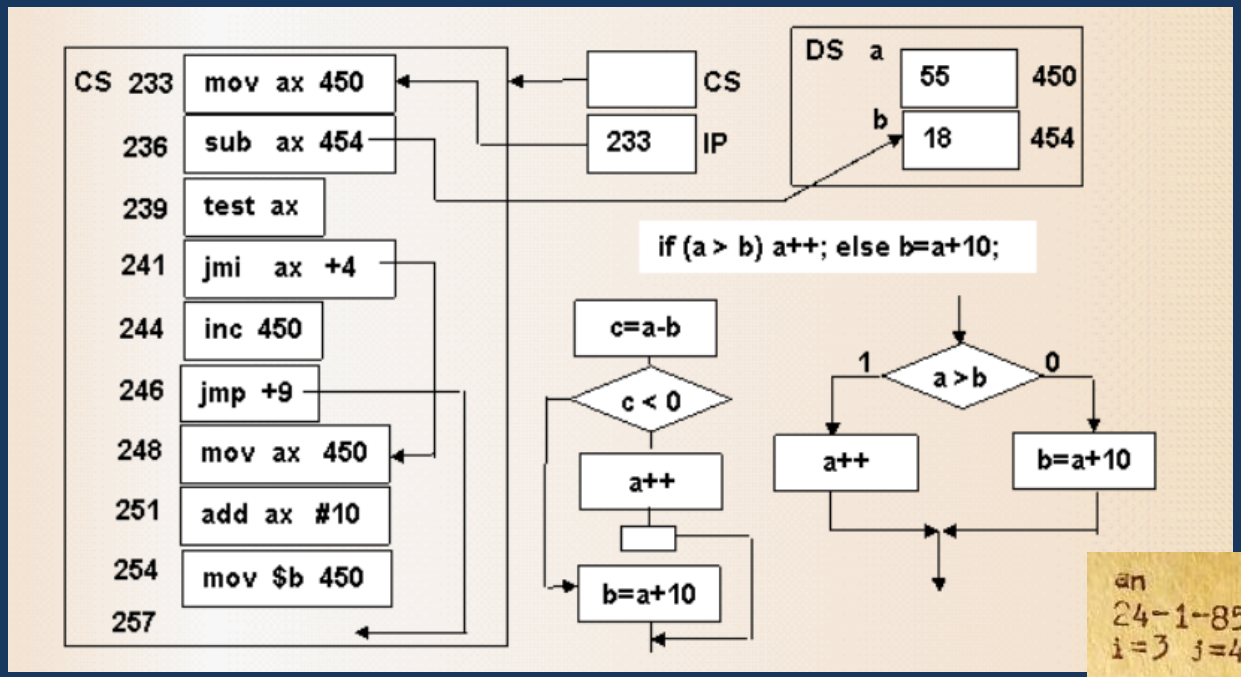
Универсальные триады групп операторов - может быть реализована *любая* логика алгоритма

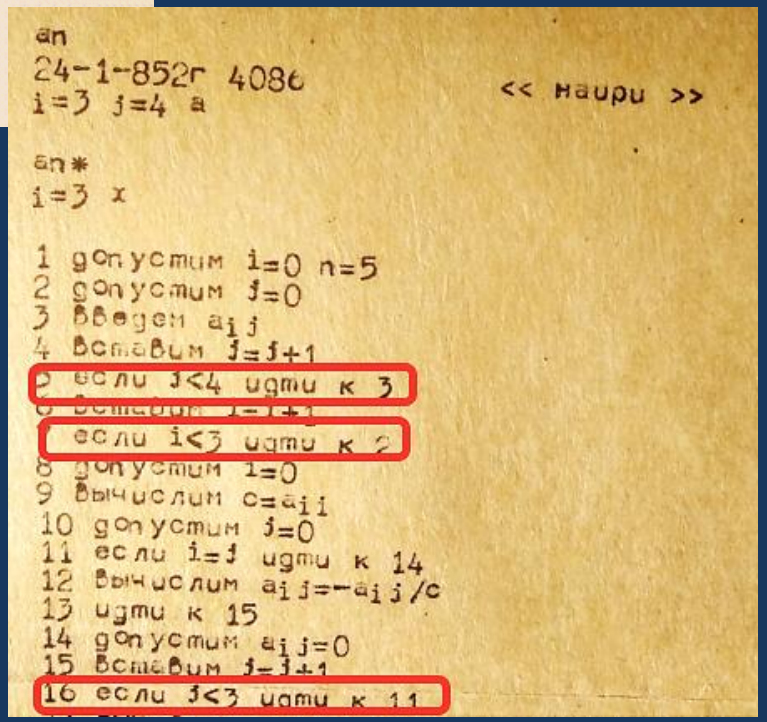
Логика алгоритма	Набор конструкций	Применение
Архитектурно-ориентированная	• действие • условие • переход	«Историческое» программирование, программирование на уровне архитектуры (системы команд) - Ассемблер
Структурированная	• последовательность • выбор (ветвление) • повторение (цикл)	Технология структурного программирования
Рекурсивная	• последовательность • выбор (ветвление) • рекурсия	Функциональное программирование (ПРОЛОГ), представление синтаксиса в формальных грамматиках (трансляторы), частично-рекурсивные функции (теория алгоритмов)

Особенности: структурированная логика алгоритма, допускающая разумные отклонения от структурного подхода

- последовательность
- ветвление
- цикл
- переход (goto, break, continue, return, switch, throw)

Архитектурно-ориентированная логика





Предпочтения:

широкое использование goto наряду с циклами

Конструкции типа **if (a>0) then goto mmm;**

Языки программирования:

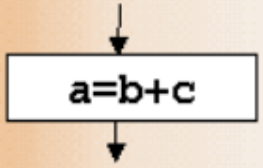
- Ассемблер
- Фортран, ранние Basic-и

Архитектурно-ориентированная логика

- Блок-схема – олицетворение архитектурно-ориентированной логики

Блок-схема содержит элементы трех видов:

- **действие**, связанное с обработкой данных, в том числе последовательность операций (выражение), присваивание и ввод-вывод. Изображается прямоугольником, имеющим один вход и один выход;



- **проверка условия.** Изображается ромбом, в котором записано условие, и который имеет один вход и два выхода в зависимости от результата (0 – ложь, 1 –истина);

- **переход** явно связывает последовательно выполняемые действия (связь по управлению, поток команд) и обозначается стрелкой.

Структурированная логика

Принцип «матрешки»:

- Синтаксическая вложенность
- Вложенность исполнения – любая конструкция имеет 1 вход – 1 выход

Синтаксические конструкции:

- последовательность
  - *неявное правило здравого смысла* – последовательно записанные конструкции одного уровня вложенности исполняются *последовательно*
  - *блок* (операторные скобки {…}) - последовательно записанные конструкции одного уровня вложенности объединяются синтаксически в одно целое (оператор)
- ветвление – if…then… else
- повторение (цикл)
- операторы перехода *запрещены,* ибо нарушают принцип «матрешки» передача управления из одного контекста (окружения) в другой (подробнее CPROG 3.3.) .

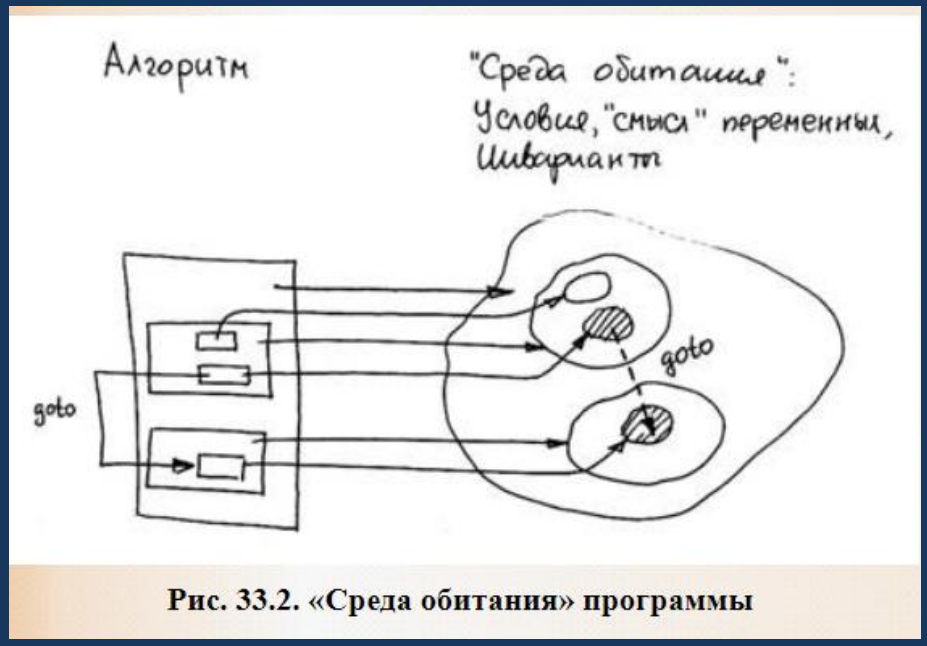
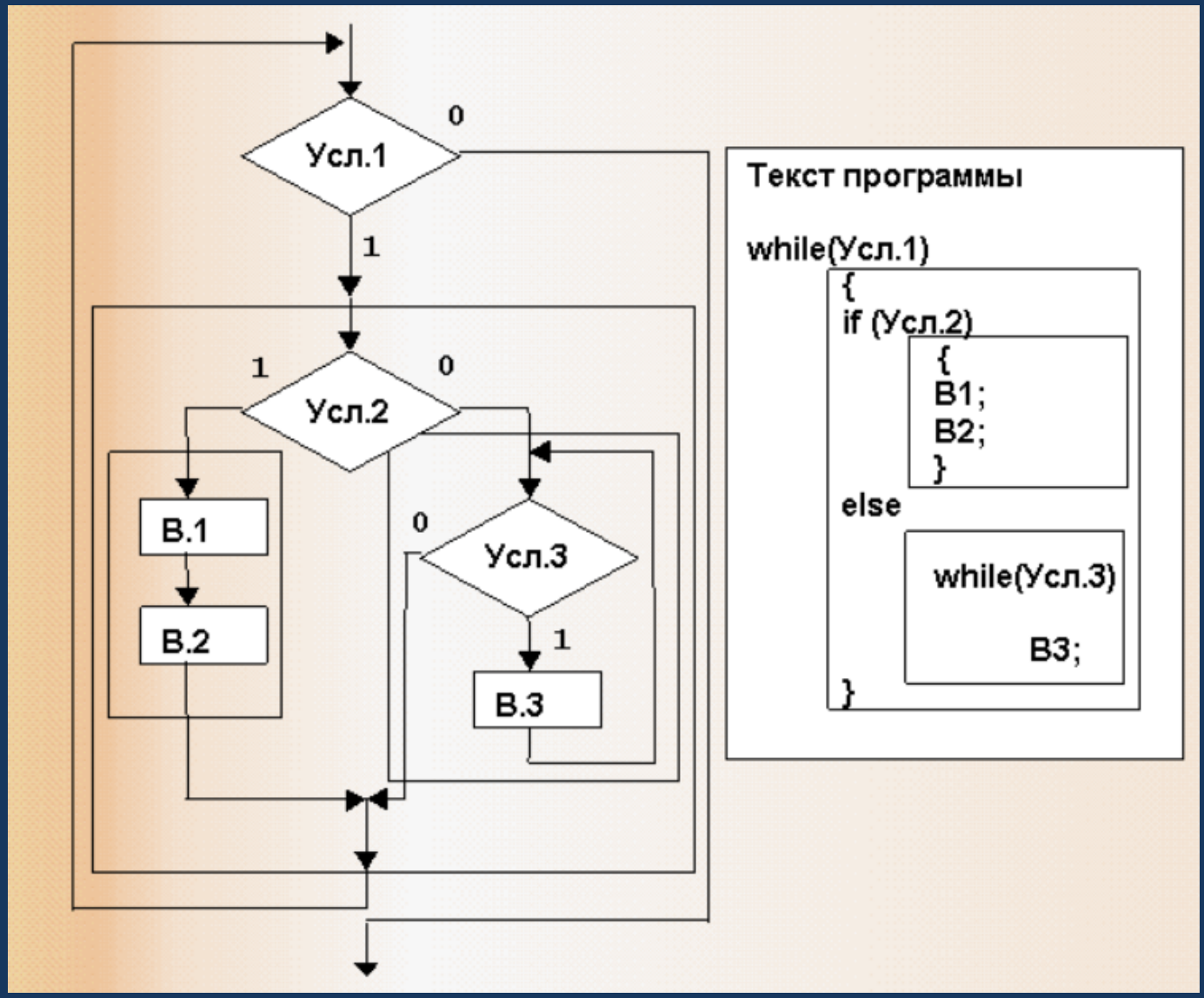


Рис. 33.2. «Среда обитания» программы

Структурированная логика

Структурированная логика и структурированная блок-схема



Последовательность действий

Неявное правило здравого смысла – последовательно записанные конструкции (операторы) одного уровня вложенности исполняются *последовательно*

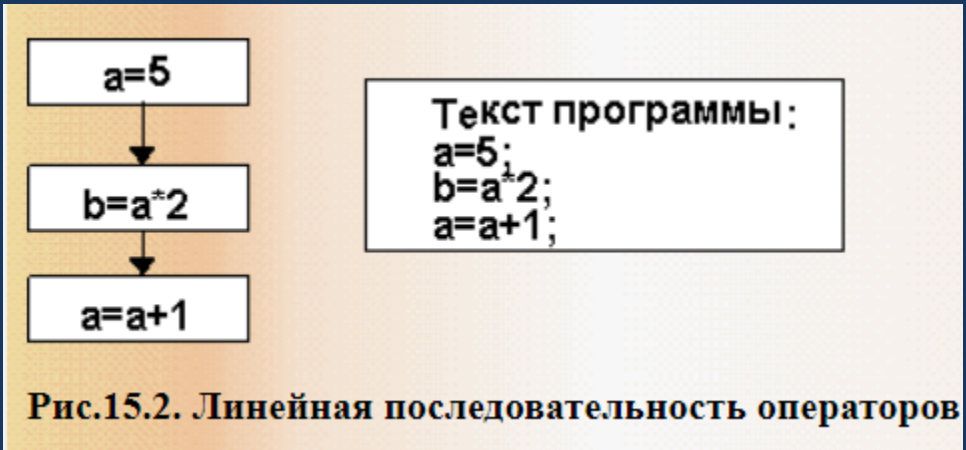


Рис.15.2. Линейная последовательность операторов

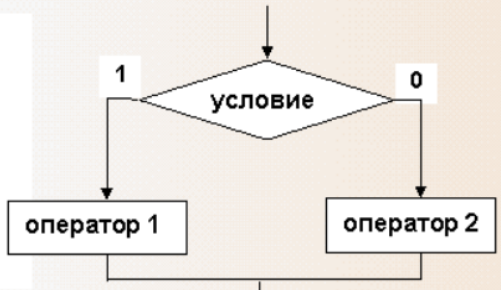
Что является оператором:

- пустой оператор - **;**
- выражение, ограниченное «;», становится оператором (другие языки – простой оператор)
- последовательность операторов, заключенная в {…}
- условный оператор
- операторы цикла
- операторы перехода

Условный оператор

```
if (условие)
        оператор 1
else
        оператор 2

if (условие) оператор
```



switch –переключатель

- Синтаксически громоздкая конструкция – сочетание if и goto

```
switch (выражение)
        {
case константа1: последовательность операторов_1
case константа2: последовательность операторов_2
case константа3: последовательность операторов_3
default:         последовательность операторов
        }
```

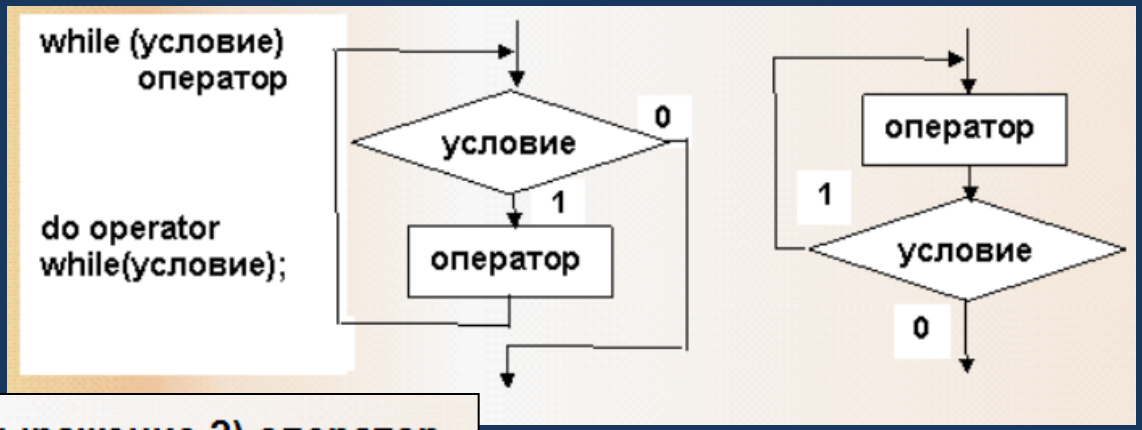
```
switch (n){                 //        Эквивалент
                            //        if (n==1) goto m1;
                            //        if (n==2) goto m2;
                              //        if (n==4) goto m3;
                              //        goto mdef;
case 1:  n=n+2; break;  // m1:   n=n+2; goto mend;
case 2:  n=0;   break;  // m2:   n=0;   goto mend;
case 4:  n++;   break;  // m4:   n++;   goto mend;
default: n=-1;          // mdef: n=-1;
        }               // mend: ...
```

- break – завершает ветку, при отсутствии выполняется последовательность для следующего case (в отличие от Basic)
- Возможная реализация в компиляторе – цикл + таблица (массив) констант + таблица меток (массив адресов перехода)

Операторы цикла

Особенности:

- во всех заголовках – условие продолжения
- переменная цикла как элемент синтаксиса отсутствует, могут использоваться любые выражения
- переменные, изменяемые в цикле, могут использоваться после его завершения (паттерн – результат цикла = место его остановки)




```
for (выражение 1; условие;  выражение 2) оператор

выражение 1;
while (условие) { оператор выражение 2; }
```

Операторы цикла

```
for (выражение 1; условие;  выражение 2) оператор

выражение 1;
while (условие) { оператор выражение 2; }
```

Цикл **for** включает в себя четыре обязательные компоненты каждого цикла: начало, условие продолжения, переход к след. шагу, тело = действие

- **выражение_1** однократно вычисляется перед началом цикла и устанавливает его начальное состояние;
- **условие** является условием продолжения цикла. Оно проверяется перед каждым шагом цикла, и при его истинности цикл повторяется. В соответствии с соглашениями, принятыми в Си, в качестве условия может выступать любое выражение со значением 0 – «ложь», не 0 – «истина»;
- на каждом шаге цикла выполняется оператор (**тело цикла**) и **выражение_2**, оба они являются повторяющейся частью цикла. Выражение_2 включает в себя необходимые действия для перехода к следующему шагу.

Замечание: как такового понятия «переменная цикла» нет. Если переменная определена вне цикла, то после его завершения она сохраняет последнее присвоенное значение

Операторы цикла

Варианты тема цикла:

1. пустой оператор – символ «**;** - точка с запятой»;
2. простой (первичный) оператор – выражение, ограниченное символом «;»;
3. единственный оператор, имеющий произвольную внутреннюю структуру своего тела – условный, цикл, переключатель;
4. составной оператор – блок, содержащий последовательность операторов, объединенную скобками «**{}**»

Примеры неочевидных циклов

```
//-----------------------------------------------------72-02.cpp
//-------"Быстрая" сортировка
 void sort(int in[], int a, int b){
 int i,j,mode;
 if (a>=b) return;                                  // Размер части =0
for (i=a, j=b, mode=1; i < j; mode >0 ? j-- : i++)  // Сокращение слева или справа
          if (in[i] > in[j]) {                      // Очередной не на своем месте
                    int c = in[i]; in[i] = in[j]; in[j]=c;   // Перестановка медианы с концевым
                    mode = -mode;                   // элементом со сменой сокращаемого конца
                    }
 sort(in,a,i-1); sort(in,i+1,b); }                   // рекурсия для частей БЕЗ медианы
```

```
//------- Сортировка массива рекурсивным разделением
// В качестве медианы - среднее арифметическое
 void sort(int in[], int a, int b){
 int i,j,mode;
 double sr=0;
 if (a>=b) return;                                  // Размер части =0
 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; }         // Слева - меньше, пропустить
          if (in[j]>=sr) { j--; continue; }         // Справа - больше, пропустить
          int c = in[i]; in[i] = in[j]; in[j]=c;
          i++,j--;                                  // Поменять местами и пропустить
          }
 if (i==a) return;                                  // все равны и в правой части
 sort(in,a,j); sort(in,i,b);}                       // рекурсивный вызов для двух частей
```

Группа операторов перехода

Операторы перехода - меняют «естественную» последовательность исполнения команд (операторов) в потоке управления:

- **goto** – безусловный переход к точке, обозначенной меткой вида «имя:»
- **continue** – переход «к *следующему* шагу цикла» (для for – выполнение части «i++» и проверке условия продолжения, do,while - проверка условия продолжения)
- **break** – досрочное завершение цикла и переход к следующему за ним оператору
- **return** – досрочное завершение исполнения (вызова) функции и возврат значение в точку вызова
- **throw** (C++) – выполнение последовательности return («разматывание стека») в поисках кода, заключенного в секцию try…catch

continue, break, return – «ограниченно нарушают» структурированный код в рамках текущей конструкции (цикл, функция), для них возможны менее компактные структурированные эквиваленты с использованием переменных-признаков

```
void      F() {                    // Не совсем синтаксически корректная
for (i=0; i<n; m1: i++)            // иллюстрация выполнения continue, break, return
          {
          if (A[i]==0)  continue;   // goto m1;
          if (A[i]==-1) return;     // goto m2;
          if (A[i] <0)   break;     // goto m3;
          }
m2:       // продолжение тела функции
          ...
m3:       // завершение функции
```

Группа операторов перехода

Допустимые случай использования goto - чрезвычайные обстоятельства, глобальные нарушения логики выполнения программы, например грубые неисправимые ошибки во входных данных. В таких случаях делается переход из нескольких вложенных конструкций либо в конец программы, либо к повторению некоторой ее части. Пример использования goto для выхода из двойного цикла (break выводит из внутреннего, необходим дополнительный признак).

Альтернативное решение – ФУНКЦИЯ с оператором return

```
retry:
int s=0;
for (int i=0;i<n;i++){
    for(int j=0;j<;j++){
    if (A[i][j]==0){
        A[i][j]=1;
        goto retry;
        }
    if (A[i][j]<0){
        A[i][j]=1;
        goto fatal;
        }
    s+=A[i][j];
    }
}
printf("s=%d\n",s);
return;
fatal:
    printf("Неудача");
}
```

«Подводные камни» лайфхаки

Варианты тела цикла/условия: пустой оператор, оператор, блок

- лишняя «;» отсекает тело цикла от заголовка – отдельный оператор ПОСЛЕ пустого цикла, синтаксически допустимая конструкция

```
int a=0;
if (a>0); a++;          // a++ после if БЕЗУСЛОВНО
int s=0;
for(int i=0;i<10;i++);
    s += A[i];          // s = A[10]
```

- При добавлении к телу цикла, состоящему из одного оператора, второго оператора, забывают создать блок {…}

```
int s1=0,s2=0;
for(int i=0;i<10;i++)
    s1 += A[i];          // в теле цикла
    s2 += A[i]*A[i];     // после цикла
```

- если условие завершения цикла сложно записать одним выражением, сделать break внутри и while(1)

- если в цикле неравномерное движение, то i++ делается не в заголовке, а в теле «по месту»

```
for(int i=0;i<n;){
    if (A[i]>0)
            i++;
    else{
        for (int k=i;k<n-1;k++)
            A[k]=A[k+1];
        n--;                // повторить после удаления
        }                   // для той же позиции
    }
}
```

Контрольные вопросы

1. Почему 4 вида управляющих конструкций являются избыточными ?
2. Перечислите возможные триады видов операторов
3. Что будет, если после заголовка цикла for поставить «;»
4. Почему после ввода заголовка цикла рекомендуется сразу ставить пару скобок {}
5. В чем разница между оператором присваивания и операцией присваивания ?
6. В каких случаях при проектировании цикла можно использовать while(1){…}
7. Что будет делать программа: int main(){ while(1); }
8. В двойном цикле for…for во внутреннем цикле произошла ошибка. Можно ли выйти из двойного цикла обычным break? Если нет, предложите решение.
9. Значение переменной после выполнения фрагмента:
   int a=1; if (a!=1); a++;