Основы конструирования программ

Преподаватель:

к.т.н., доцент кафедры Инженерной психологии и эргономики
Меженная Марина Михайловна
mezhennaya@bsuir.by
а 606-2

Структура дисциплины «Основы конструирования программ»

1 семестр: лекции + практические занятия → зачет

2 семестр: курсовая работа

Лекция 1: Жизненный цикл разработки программного обеспечения (ПО). Конструирование ПО. Основы теории алгоритмов.

План лекции:

1.Жизненный цикл разработки ПО. Конструирование ПО.

2.Алгоритмические и неалгоритмические стили и языки программирования.

3.Алгоритм: определение, свойства, способы описания, базовые конструкции.

4.Принципы организации последовательного кода, управляющих структур (условные операторы, циклы).

5.C++ Code Convention.

Жизненный цикл разработки ПО:

1.Идея создания ПО (постановка задачи)

2.Разработка требований

3.Проектирование:

- разработка архитектуры (высокоуровневое проектирование)
- детальное проектирование

4.Реализация (кодирование)

5.Тестирование

6.Сопровождение

Конструирование ПО = детальное проектирование + программирование

Детальное проектирование – декомпозиция системы до уровня очевидно реализуемых модулей или процедур.

Программирование в части:

- проектирование классов и методов;
- создание имен переменных и констант;
- выбор управляющих структур и организация блоков команд;
- «шлифовка» кода путем его тщательного форматирования и комментирования;
- оптимизация кода, направленная на повышение его быстродействия, и снижение степени использования ресурсов;
- интеграция программных компонентов, созданных по отдельности.

Алгоритмические и неалгоритмические языки программирования

Алгоритмические языки программирования (императивные): описывается алгоритм решения задачи.

Примеры: Pascal, Delphi, C, C++, Java.

Неалгоритмические языки программирования (декларативные): описывается результат (его свойства), а не методы его достижения.

Примеры: Prolog, Lisp, Mercury.

Алгоритмические языки программирования

Предметом данного курса является конструирование алгоритмических программ.

Такие программы используют алгоритмическую организацию программного кода, а потому базовым понятием при их конструировании является **алгоритм.**

При этом процесс конструирования может быть представлен в следующем виде:
анализ задачи → разработка алгоритма → программирование.

Конструирование ПО: анализ задачи

Анализ – это исследование объектов или явлений, путем изучения составляющих его элементов.

Анализ задачи позволяет установить входные и выходные данные, выделить основные решения между входными и выходными данными, выделить модули, необходимые для выполнения задачи, и определить методы их частных решений.

Конструирование ПО: разработка алгоритма

Алгоритм – конечная последовательность команд исполнителю для решения поставленной задачи.

Формальный исполнитель – субъект, механически реализующий алгоритм.
Совокупность допустимых действий образуют **систему команд исполнителя.**

Применительно к разработке ПО формальным исполнителем алгоритма выступает компьютер, а системой команд исполнителя является совокупность допустимых команд конкретного языка программирования.

Алгоритмизация – процесс разработки алгоритма.

Свойства алгоритма:

1.**Понятность** – каждая команда должна входить в систему команд исполнителя.

2.**Дискретность** – алгоритм должен представлять процесс решения задачи как последовательное выполнение шагов.

3.**Определенность** – каждое правило алгоритма должно быть четким, однозначным.

4.**Результативность** – при точном исполнении всех предписаний алгоритма процесс должен прекратиться за конечное число шагов и при этом должен получиться определённый результат. Вывод о том, что решения не существует - тоже результат.

5.**Массовость** – алгоритм должен решать любую задачу из того класса задач, для решения которых он разработан.

Формы представления алгоритма

1. Словесная

Пример словесной записи:

Записать алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел.

1.задать два числа;

2.если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;

3.определить большее из чисел;

4.заменить большее из чисел разностью большего и меньшего из чисел;

5.повторить алгоритм с шага 2.

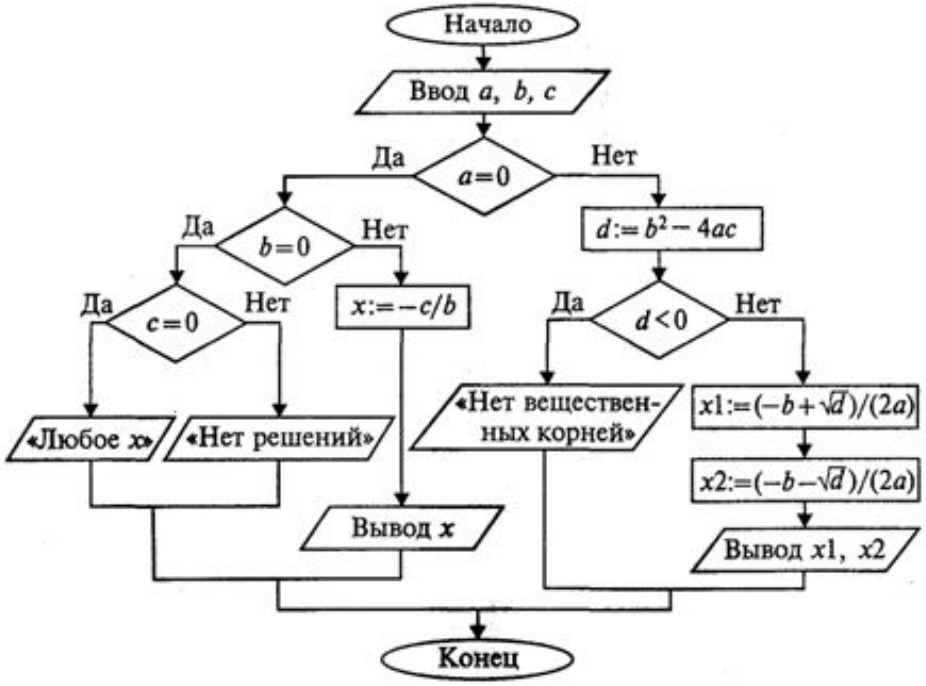
Формы представления алгоритма

2.Псевдокод (полуформализованное описание алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.)

```
процедура print_word(символный указатель w)
{
целое i = 0;
пока w[i] истинно (пока i элемент массива w существует)
вывести на экран w[i++](следующий элемент w);
}
```

Формы представления алгоритма

3. Графическая (блок-схемы)

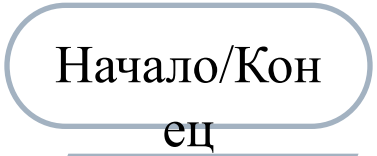


Формы представления алгоритма

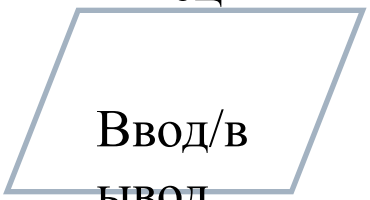
4. Программная (текст на языке программирования).

```
#include <iostream>
using namespace std;

int main()
{
   cout << "Hello, world!" << endl;
   system("pause");
return 0;
}
```

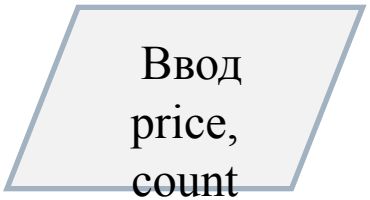
Основные символы блок-схемы алгоритма

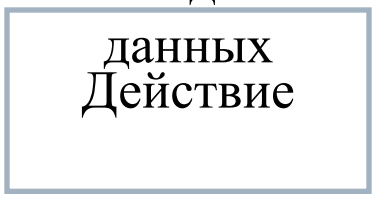


Блок начала или конца алгоритма

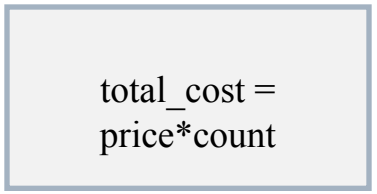


Блок ввода или вывода данных

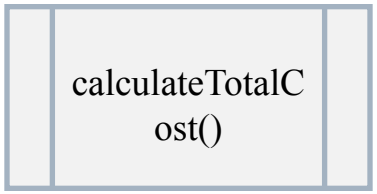




Функциональный блок



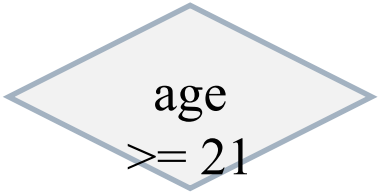
Блок вызова предопределенного процесса (процедуры/функции)

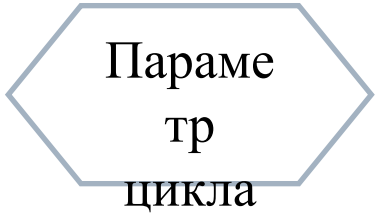


Соединительный блок

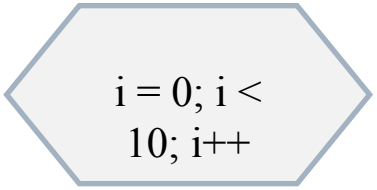
Основные символы блок-схемы алгоритма

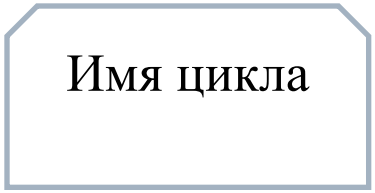
Логический блок
(для реализации
условного оператора if)



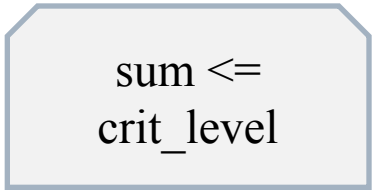


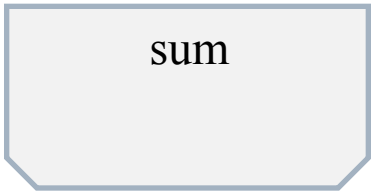
Блок модификации
(для реализации цикла for)



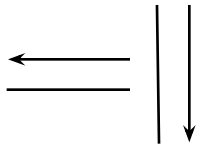


Блоки границ цикла
(для реализации циклов
while, do-while)



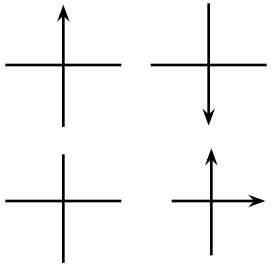


Обозначение соединений между символами блок-схемы алгоритма

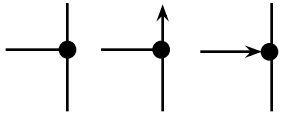


Указание направления линии потока:
допускается без стрелки, если линия направлена слева направо и сверху вниз;
со стрелкой в остальных случаях.

Изменение направление потока (под углом 90°)

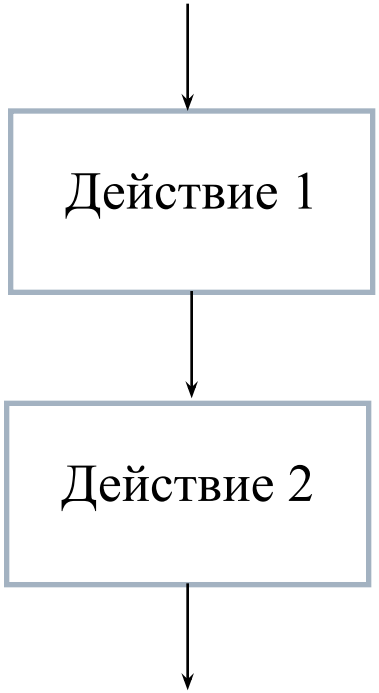


Пересечение двух несвязанных потоков

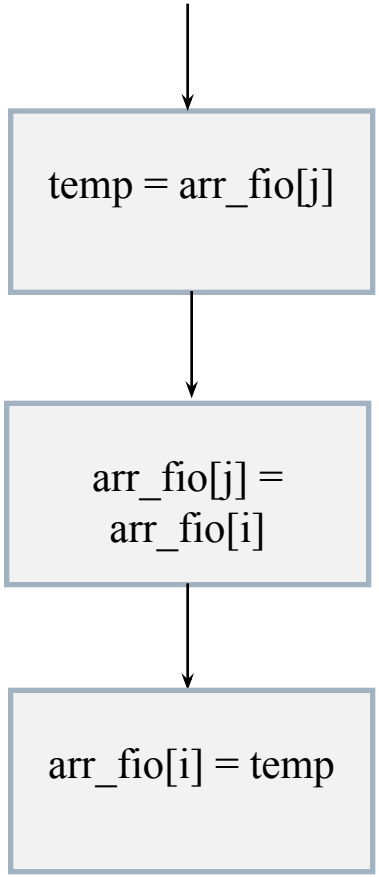


Слияние линий потока, каждая из которых направлена к одному и тому же символу на схеме

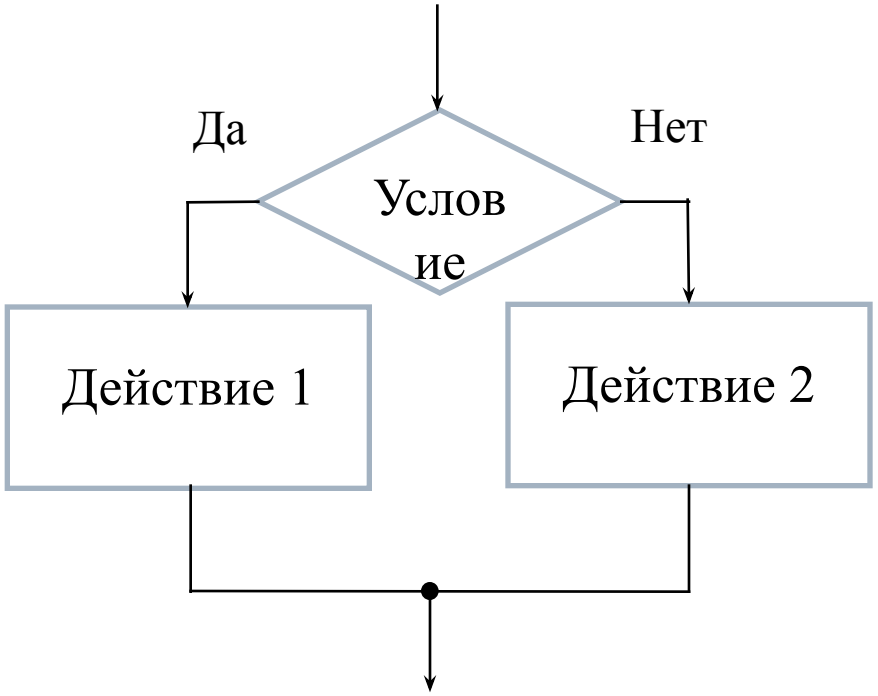
Базовые алгоритмические структуры: следование



Пример:
меняем местами два элемента массива

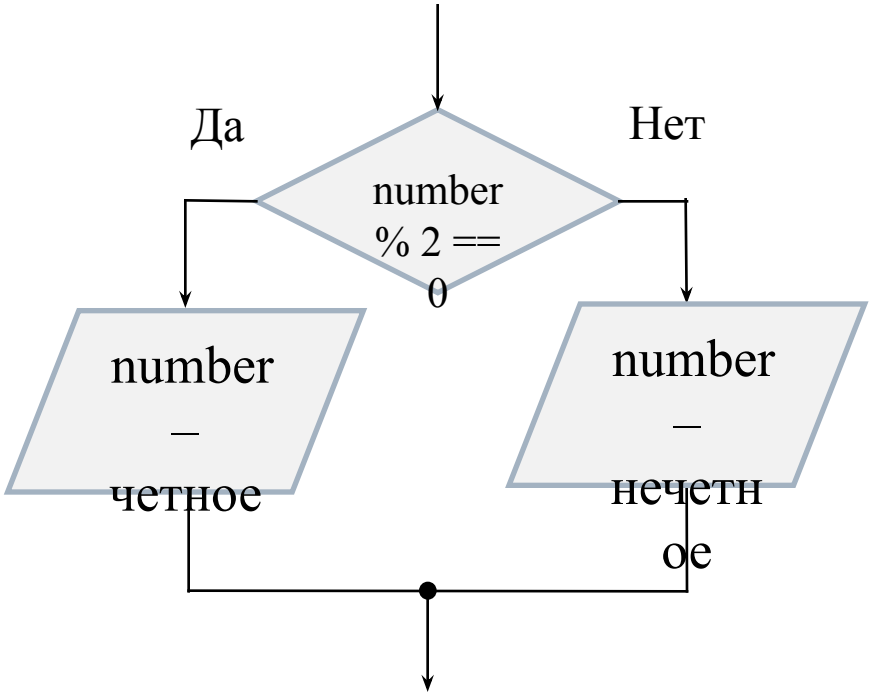


Базовые алгоритмические структуры: ветвление (полное)
Реализуется посредством оператора if-else



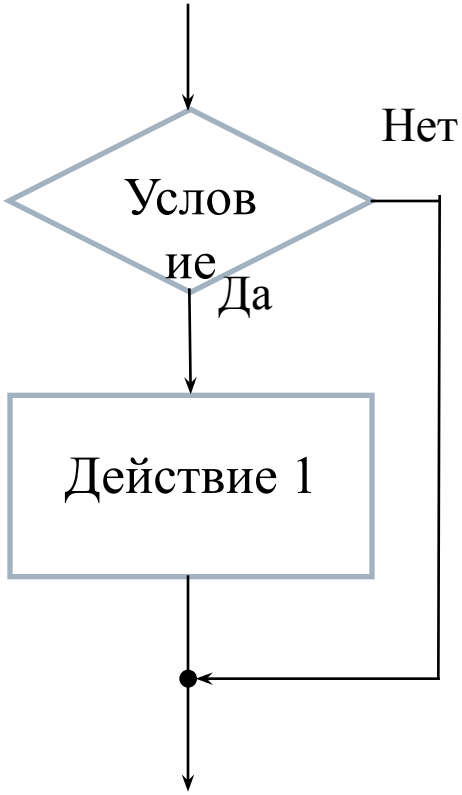
Базовые алгоритмические структуры: ветвление (полное) Реализуется посредством оператора if-else

Пример: определить четное или нечетное число.

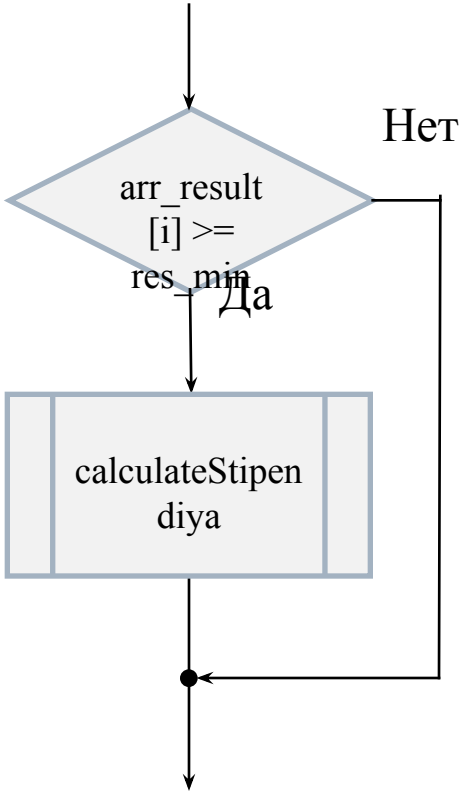



```
// проверка на четность

int number;
cout << "Enter number: " << endl;
cin >> number;

if (number % 2 == 0)
   {
   cout << "Четное" << endl;
} else
{
   cout << "Нечетное" << endl;
}
```

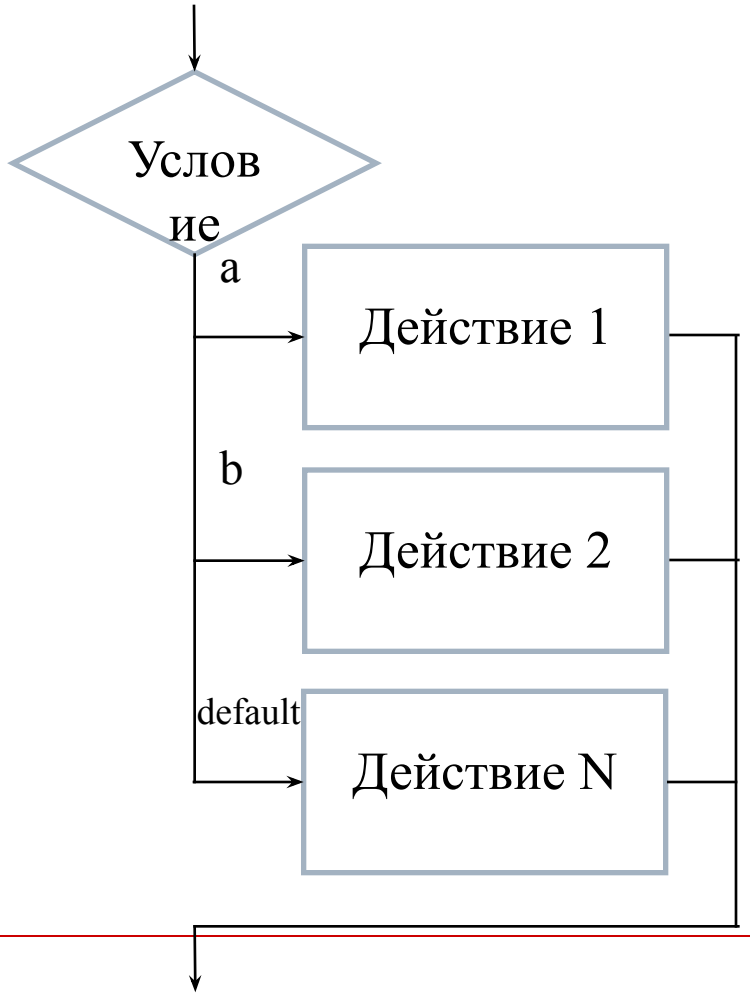
Базовые алгоритмические структуры: ветвление (неполное)
Реализуется посредством оператора if



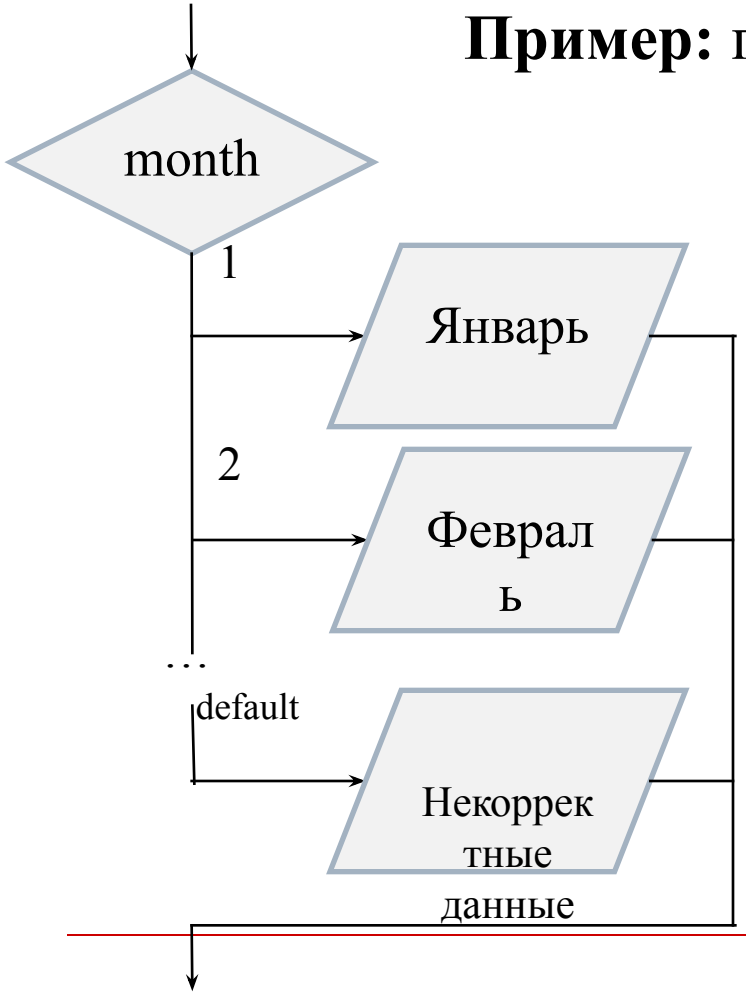
Пример: начислить стипендию, если студент набрал достаточно баллов за сессию



Базовые алгоритмические структуры: ветвление (выбор)
Реализуется посредством оператора switch-case



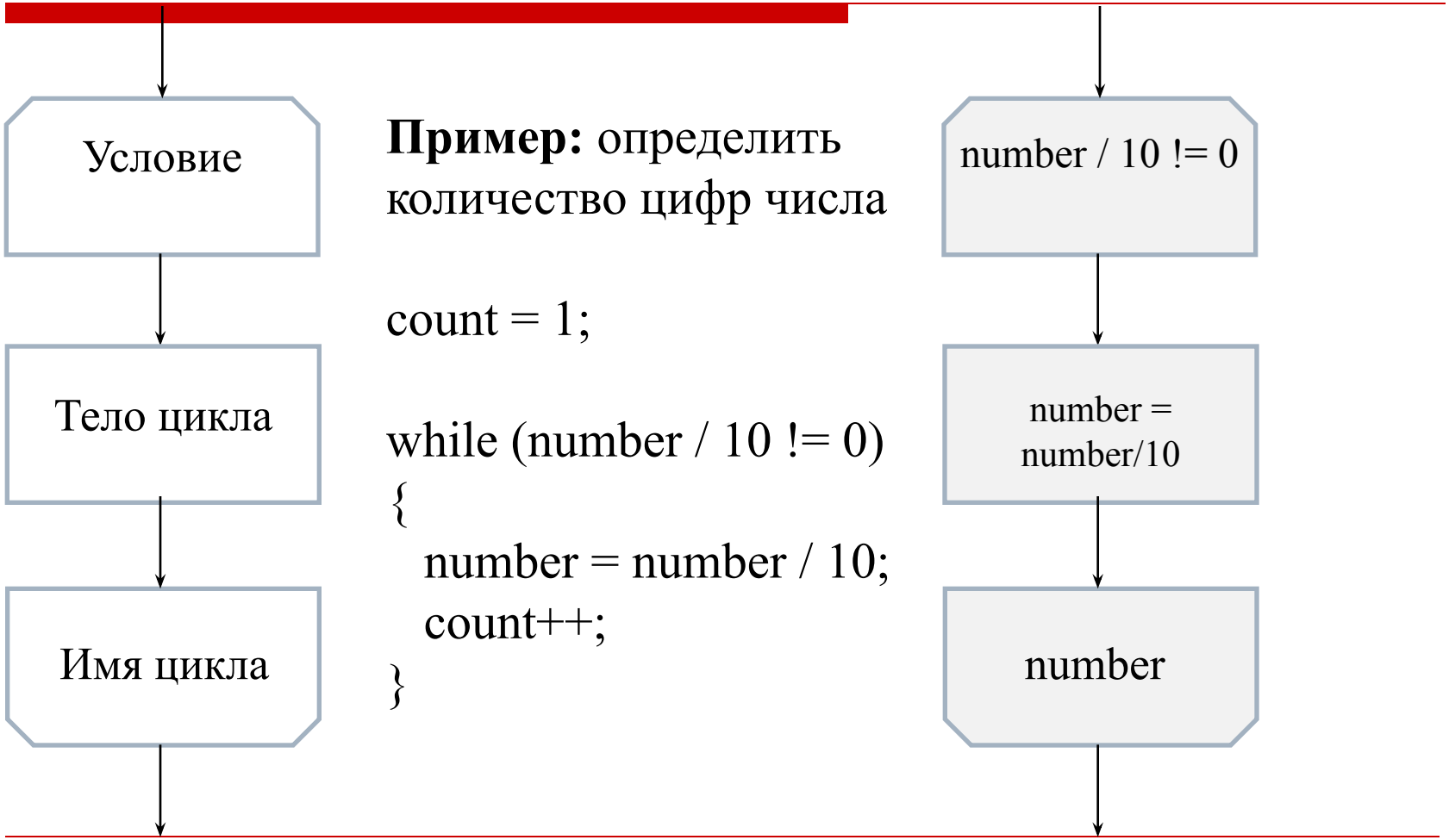
Базовые алгоритмические структуры: ветвление (выбор)
Реализуется посредством оператора switch-case



Пример: по номеру месяца вывести его название

```
switch (month) {
  case 1:
     cout << "Январь" << endl;
     break;
  case 2:
     cout << «Февраль" << endl;
     break;
  …
  case 12:
     cout << "Декабрь" << endl;
     break;
  default:
     cout << "Введите корректные данные!" <<   endl;
     break;
}
```

Базовые алгоритмические структуры: цикл (с предусловием)
Реализуется посредством оператора while



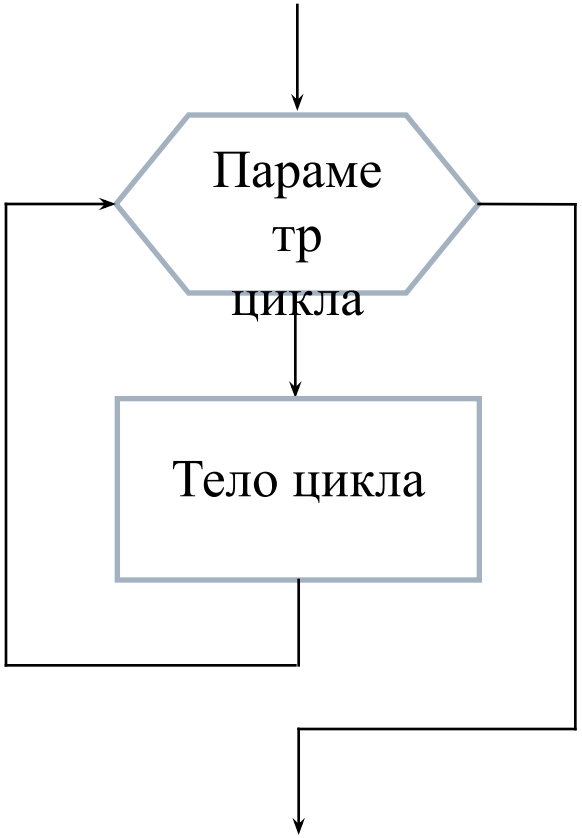
Пример: определить количество цифр числа

```
count = 1;

while (number / 10 != 0)
{
   number = number / 10;
   count++;
}
```

Базовые алгоритмические структуры: цикл (с постусловием)
Реализуется посредством оператора do-while

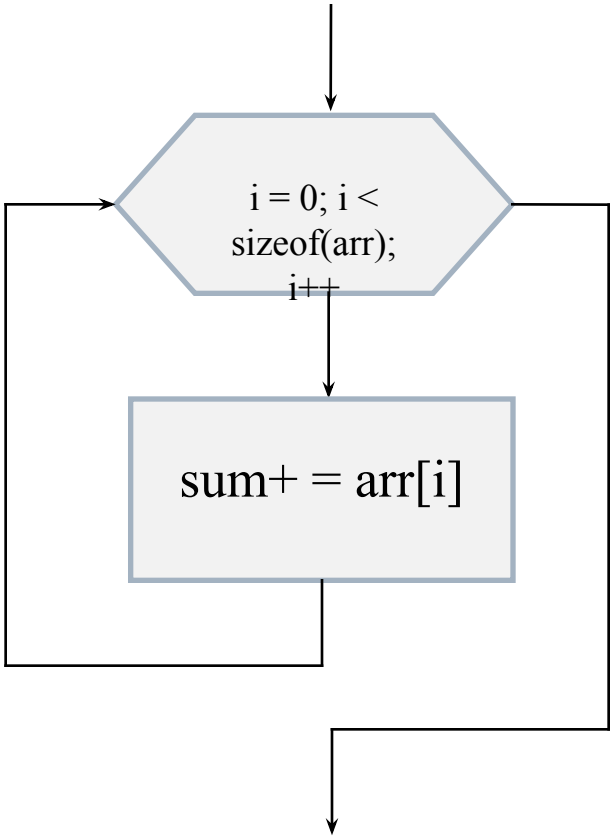
Пример:
реализовать игру в карты между пользователем и компьютером (игра продолжается, пока не найден победитель)

Базовые алгоритмические структуры: цикл (с предусловием)
Реализуется посредством оператора for



Пример: найти сумму всех элементов массива

```
int sum = 0;

for (int i = 0; i < sizeof(arr); i++)
{
   sum+ = arr[i];
}
```



Принципы организации последовательного кода

Главный принцип организации последовательного кода – **группировать взаимосвязанные выражения** в блоки и отделять их в коде **пустой строкой**. Взаимосвязанные выражения работают с одними и теми же данными, выполняют схожие задачи или зависят от порядка выполнения друг друга.

Хорошо организованный код

Неудачно организованный код

Принципы организации условных операторов
Оператор if

Размещайте **наиболее вероятные** варианты раньше остальных. Так вы увеличите эффективность, потому что уменьшите число проверок, выполняемых кодом в большинстве случаев.

Убедитесь, что при сравнении **на равенство** ветвление корректно. Использование > вместо >= или < вместо <= это риск ошибки!

Избегайте вложенности if **более трех.**

Сначала напишите код номинального хода алгоритма, затем опишите исключительные случаи.

Принципы организации условных операторов
Оператор switch

Организовать **порядок следования** вариантов можно по-разному:
-по алфавиту или численно, если все варианты равновероятны;
-правильный вариант → первый, далее – исключения;
-варианты по вероятности появления.

Сделайте обработку каждого варианта **простой**. Код, связанный с каждым вариантом, должен быть коротким. Если действия, предпринимаемые для какого-то варианта слишком сложны, напишите **метод** и вызывайте его.

Используйте вариант по умолчанию (**default**) только для обработки настоящих значений по умолчанию, но не кодируйте последний оставшийся вариант как вариант по умолчанию.

Принципы организации операторов цикла

Правила выбора вида цикла:

Если вы заранее **не знаете**, сколько итераций должен выполнить цикл, используйте **while.**

Если Вы точно знаете, что цикл должен выполняться **хотя бы раз**, то используйте вариант **do while.**

Цикл **for** – хороший вариант, если вам нужен цикл, выполняющийся **определенное количество раз**.

Используйте смысловые имена переменных, чтобы сделать вложенные циклы читабельными.

Ограничивайте вложенность тремя уровнями. Если вам нужно большее число уровней, сделайте цикл короче (концептуально), вынеся его часть в отдельный метод.

C++ Code Convention

```
1  #include<iostream>
2  using namespace std;
3  int main(){ setlocale(LC_ALL, "rus");
4  int boxWithFruit=15;//количество ящиков на складе
5  int amountBoxForSale=0;// количество отгружаемых ящиков
6  cout<<"На складе сейчас"<<boxWithFruit<<" ящиков с фруктами.\n\n";
7  for(int i=1;;i++)//i - количество машин к погрузке
8  {cout<<"Сколько ящиков отгрузить в "<<i<<"-ю машину? ";
9  cin>>amountBoxForSale;
10 if(amountBoxForSale>boxWithFruit)
11 {cout<<"\nНа складе нет такого количества товара!";
12 cout<<"Осталось только "<<boxWithFruit<<" ящиков\n\n";
13 i--;//уменьшить счетчик на 1
14 }else{boxWithFruit-=amountBoxForSale;//перезаписываем значение
15 cout<<"Осталось "<<boxWithFruit<<" ящиков.\n";
16 }if(boxWithFruit==0)//если ящиков больше нет - выйти из цикла
17 {cout<<"Фрукты закончились! До свидания!\n";
18 break;}}return 0;}
```

C++ Code Convention

```cpp
#include<iostream>
using namespace std;

int main()
{
    setlocale(LC_ALL, "rus");

    int boxWithFruit = 15; // количество ящиков на складе
    int amountBoxForSale = 0; // количество отгружаемых ящиков

    cout << "На складе сейчас" << boxWithFruit << " ящиков с фруктами.\n\n
    for (int i = 1;; i++) //  i - количество машин к погрузке
    {
        cout << "Сколько ящиков отгрузить в " << i << "-ю машину? ";
        cin >> amountBoxForSale;

        if (amountBoxForSale > boxWithFruit)
        {
            cout << "\nНа складе нет такого количества товара!";
            cout << "Осталось только " << boxWithFruit << " ящиков\n\n";
            i--; // уменьшить счетчик на 1
        }
        else
        {
            boxWithFruit -= amountBoxForSale; // перезаписываем значение
            cout << "Осталось " << boxWithFruit << " ящиков.\n";
        }

        if (boxWithFruit == 0)// если ящиков больше нет - выйти из цикла
        {
            cout << "Фрукты закончились! До свидания!\n";
            break;
        }
    }
    return 0;
}
```

C++ Code Convention:
Имена файлов, переменных, констант, функций, классов

Имя файла должны состоять только из букв нижнего регистра и цифр, допускается нижнее подчеркивание (**io_base.cpp, server1.c**)

Имя константы – имя существительное, должно состоять из букв верхнего регистра (**HOURS_IN_DAY, SIZE**)

Имя переменной – имя существительное, всегда начинается с буквы нижнего регистра, допускается нижнее подчеркивание (**name, age; boxWithApple, box_with_apple**)

Имя класса – имя существительное, всегда начинается с буквы верхнего регистра (**class User; class MyAdapter**)

Имя функции (метода) – начинается с глагола, всегда начинается с буквы нижнего регистра (**printData(), setName(); showStr()**)

C++ Code Convention
Имена файлов, переменных, констант, функций, классов

Главный принцип – дать переменной (функции, классу и т.д.) **осмысленное имя**, как можно более близкое к контексту использования:
age – возраст;
number – номер;
amount – количество;
name – имя.

Желательно имена писать не английским транслитом, а английскими словами:
не vozrast – a age;
не kolichestvo – a amount.

C++ Code Convention
Фигурные скобки

Рекомендовано использовать фигурные скобки в блоках if, else, while, do, for даже если они содержат всего одну строку. Например:

```
1 for(int example = 0; example < 10; example ++)
2 {
3     cout << example << end;
4 }
```

Каждую фигурную скобку желательно располагать в отдельной строке. Так очень легко проследить, где блок начинается и где заканчивается.

C++ Code Convention
Пробелы в строке

При использовании оператора присвоения значения и операторов арифметических операций пробелы необходимы с обоих сторон от этого оператора:

```cpp
variable = a + b - c;
variable=a+b-c; // слитно выглядит так
```

```cpp
variable = -4;

variable++;
```

Исключение – унарные операторы

C++ Code Convention
Табуляция

Без табуляции

```
1  int main()
2  {
3  for (int i = 0; i < 12; i++)
4  {
5  for (int j = 0; j < 12; j++)
6  {
7  cout << '@';
8  }
9  cout << endl;
10 }
11 return 0;
12 }
```

С табуляцией

```
1  int main()
2  {
3          for (int i = 0; i < 12; i++)
4          {
5                  for (int j = 0; j < 12; j++)
6                  {
7                          cout << '@';
8                  }
9                  cout << endl;
10         }
11         return 0;
12 }
```

C++ Code Convention

Комментарии играют важную роль в поддержании читаемости кода!

Оставлять комментарии в коде можно либо используя двойной слэш // (комментирование одной строки), либо /* комментарий */ (многострочный комментарий).

```
/*
int number;                                  // декларация переменной
cout << "Enter number: " << endl;     // вводим числа с клавиатуры
cin >> number;                              // запись введенного числа в переменную number
*/
```

C++ Code Convention

В Microsoft Visual Studio есть “спасательная комбинация клавиш” **Ctrl+K затем Ctrl+F**, нажав которую осуществится **форматирование** выделенного исходного кода.