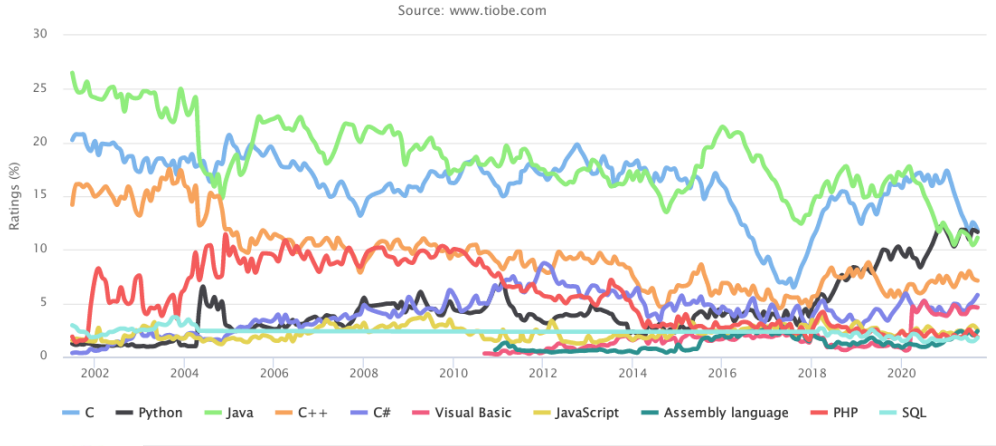
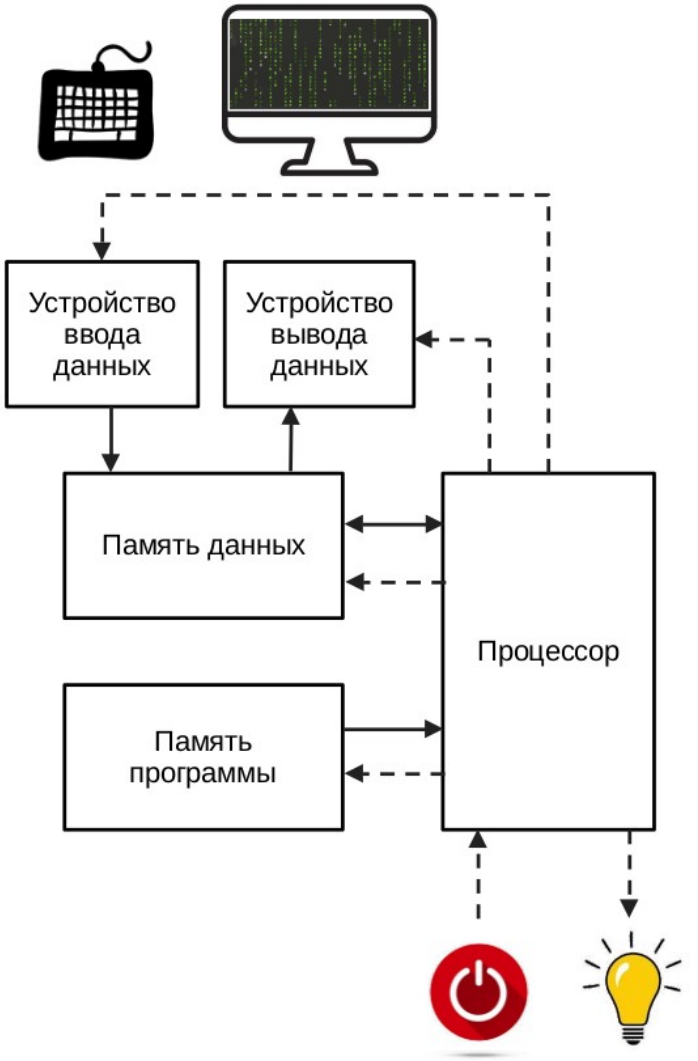
Способы задания однозначности в архитектурах BC. Связь с типизацией



Начальный взгляд

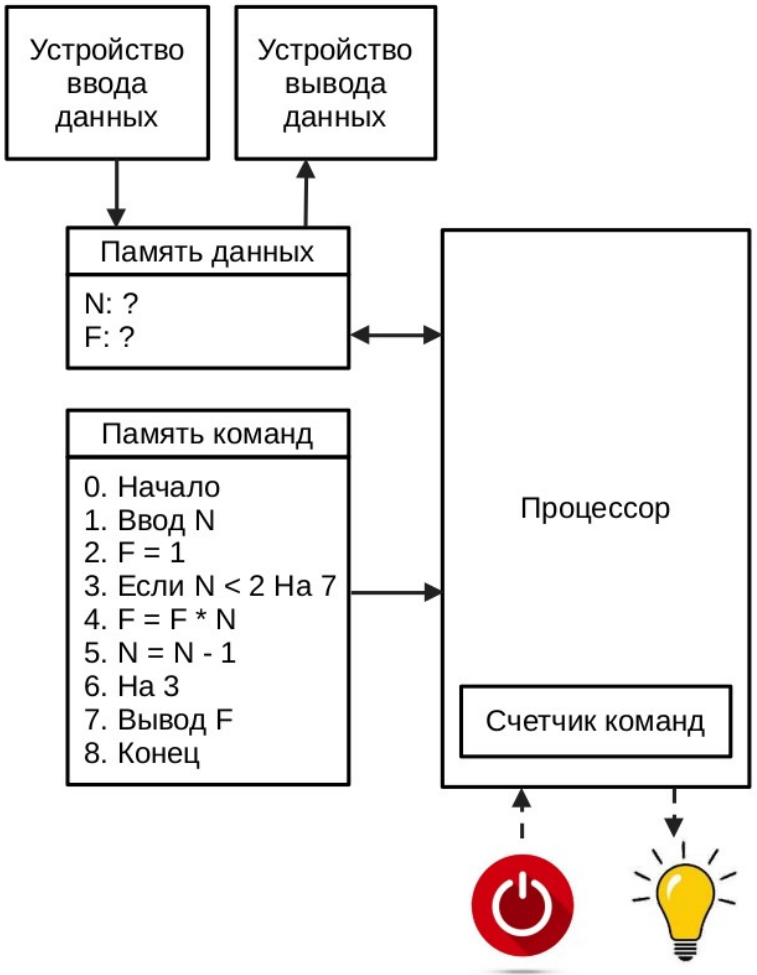
1. Начало
2. Конец
3. Ввод
4. Вывод
5. Операция
6. Условный переход
7. Безусловный переход

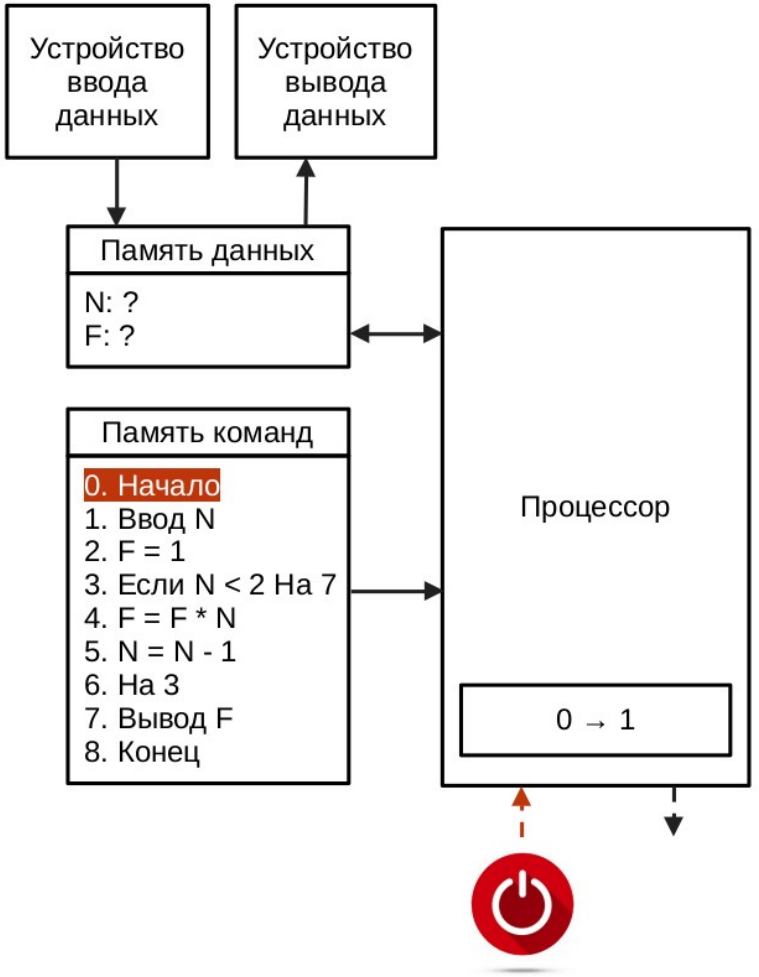


Что внутри?

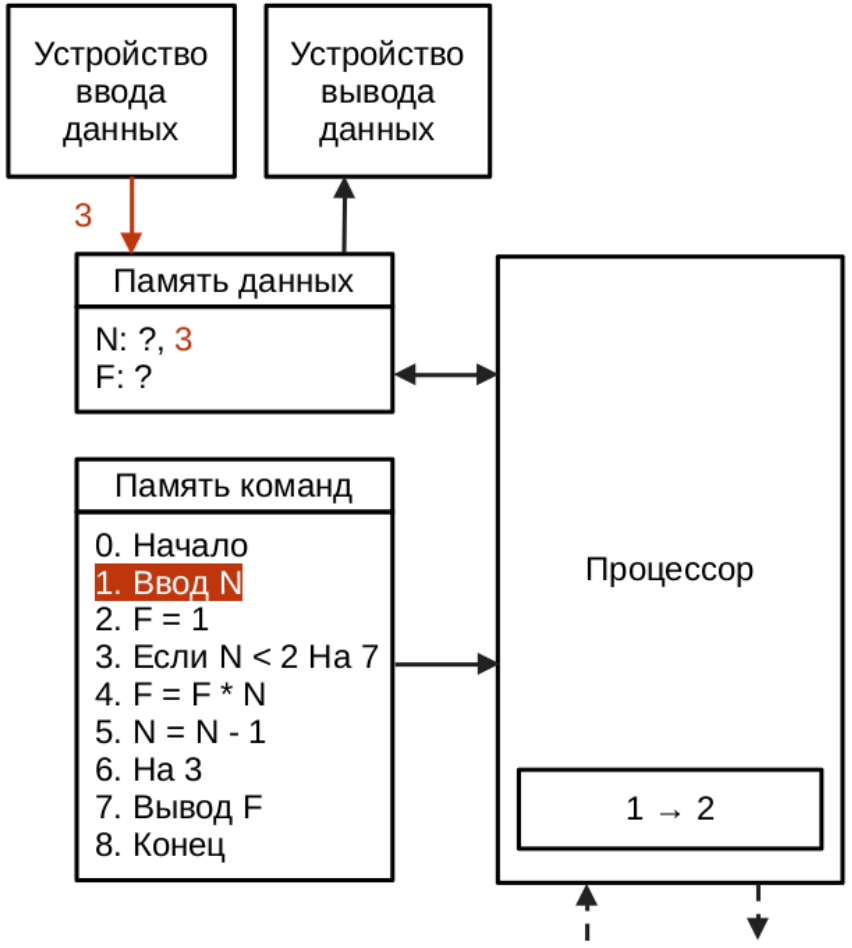
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец

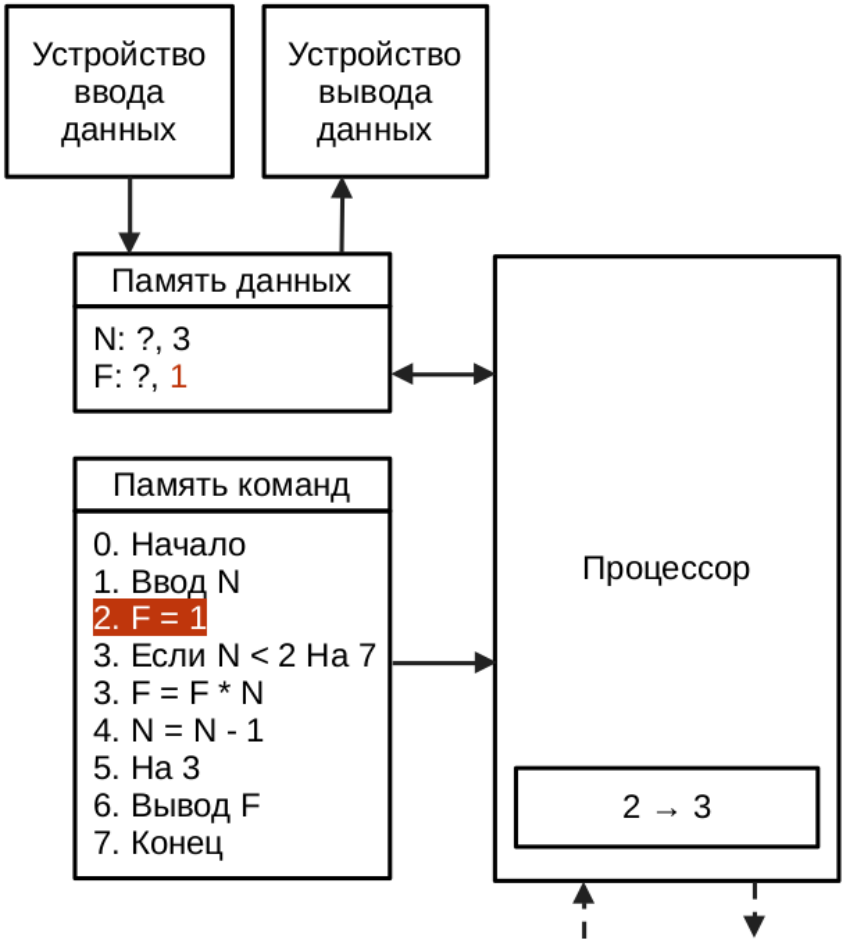
Что внутри?


Устройство ввода данных
Устройство вывода данных
Память данных
N: ?
F: ?
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
Счетчик команд

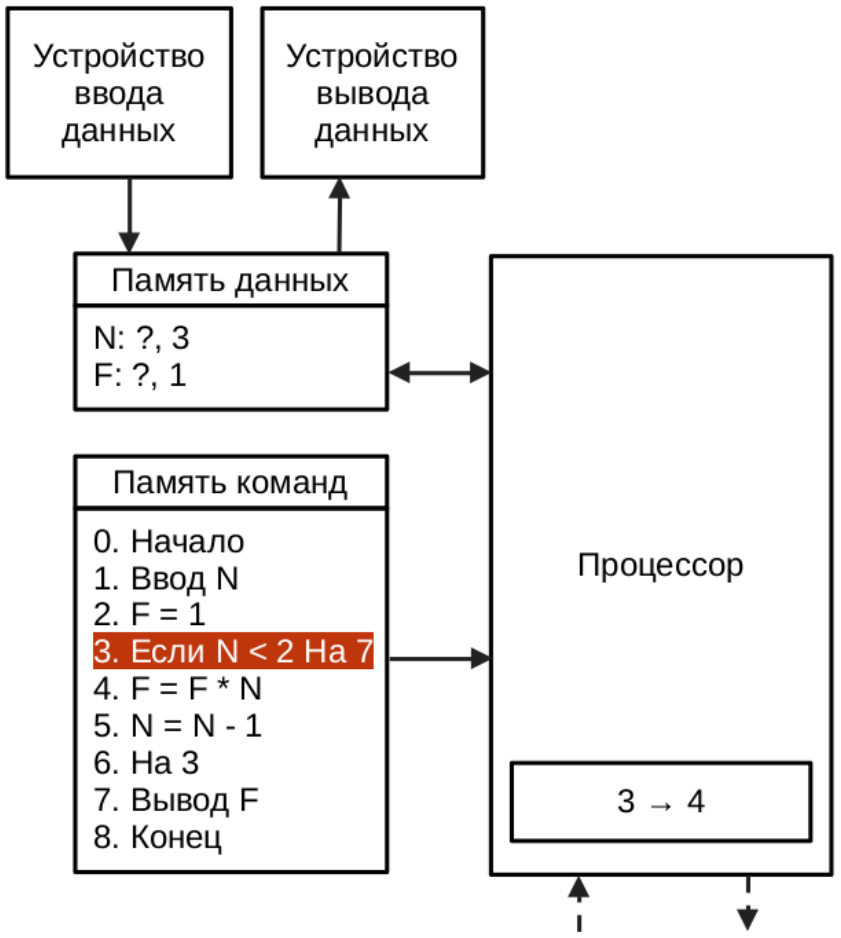

Устройство ввода данных
Устройство вывода данных
Память данных
N: ?
F: ?
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
0 → 1

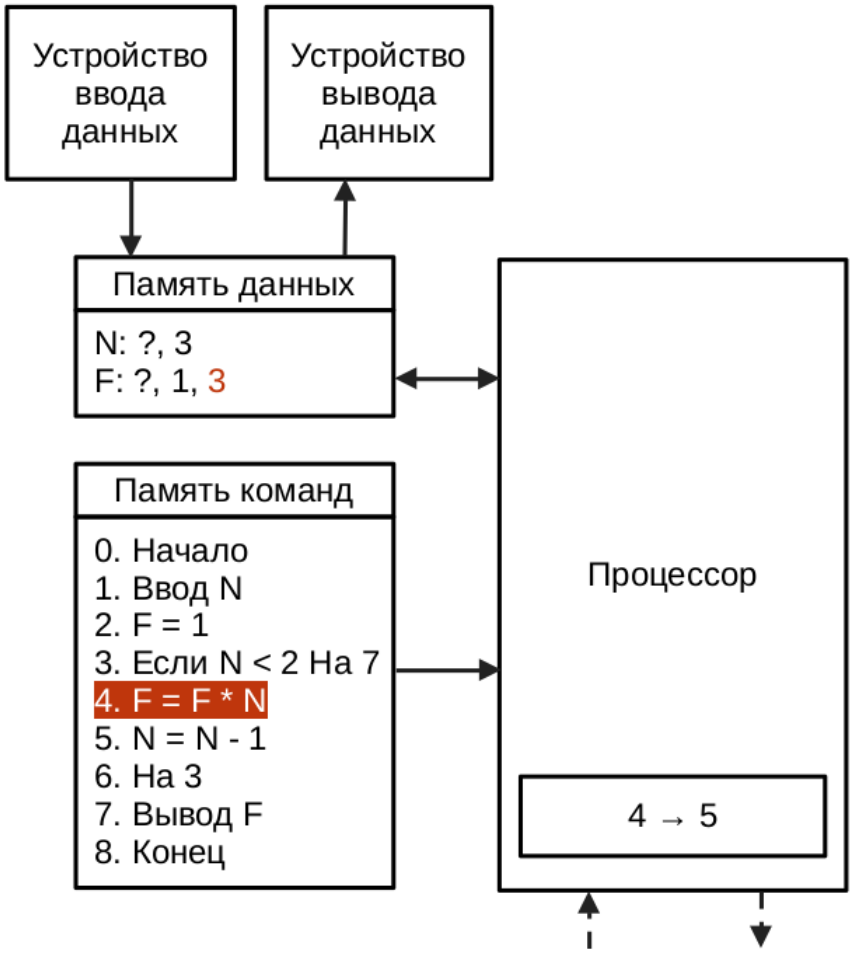
Что внутри?


Устройство ввода данных
Устройство вывода данных
3
Память данных
N: ?, 3
F: ?
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
1 → 2

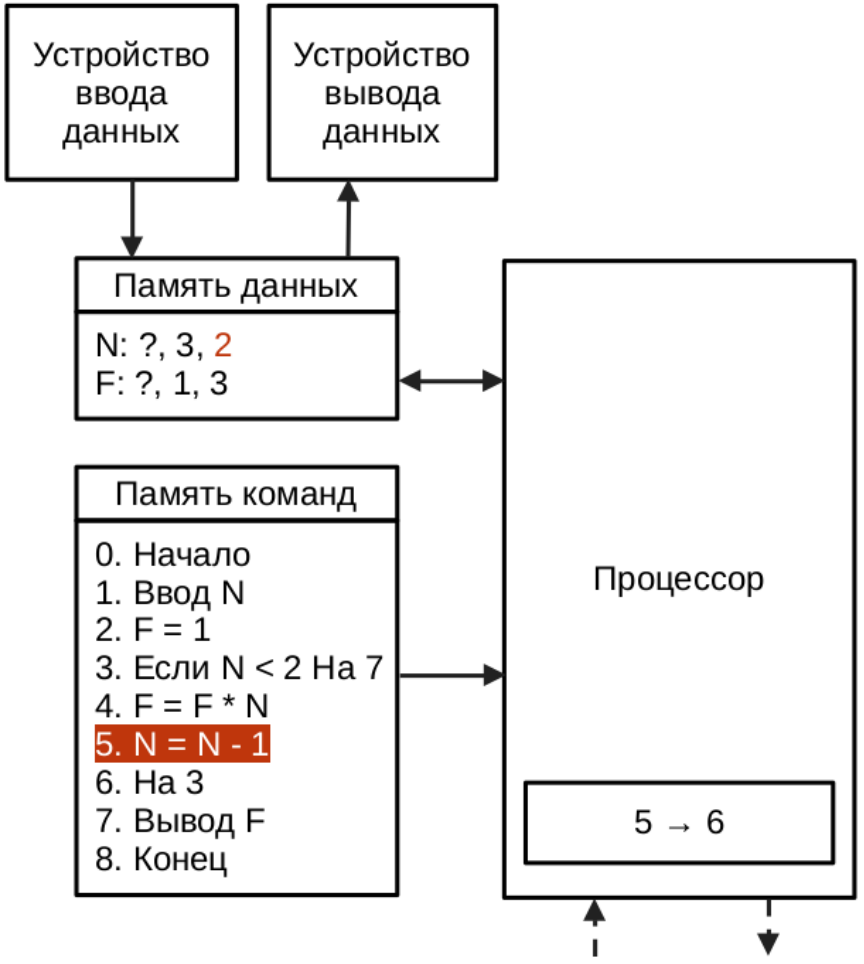

Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3
F: ?, 1
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
3. F = F * N
4. N = N - 1
5. На 3
6. Вывод F
7. Конец
Процессор
2 → 3

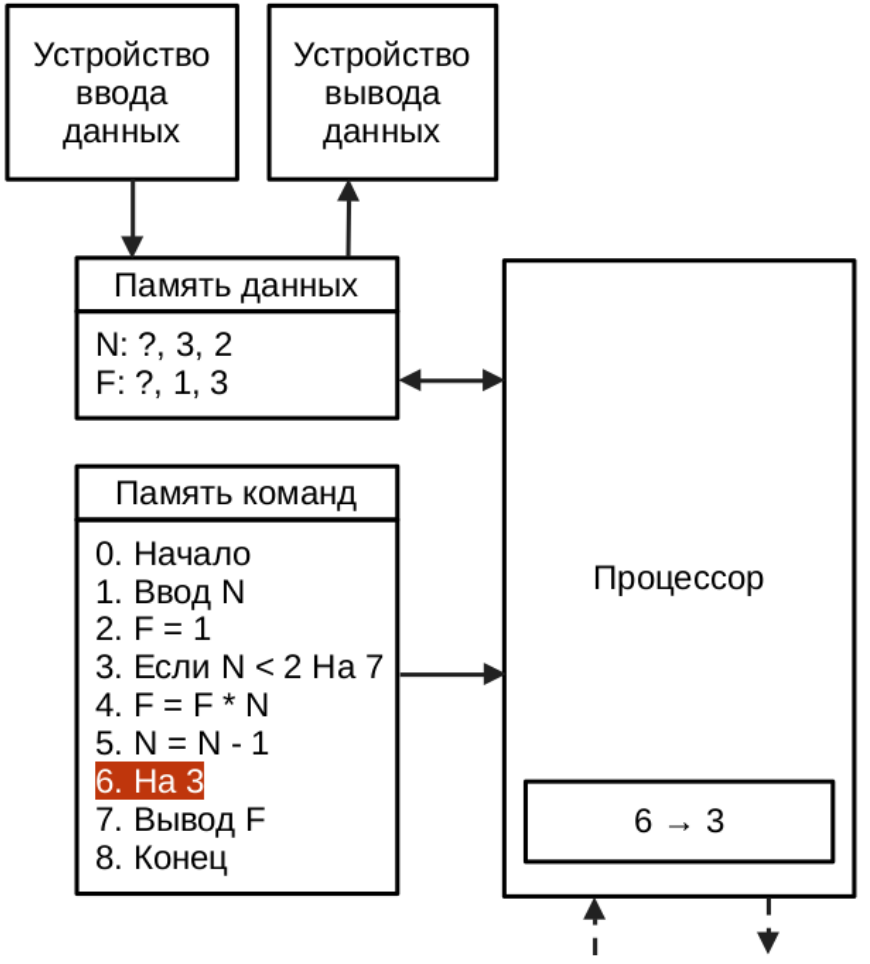
Что внутри?


Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3
F: ?, 1
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
3 → 4

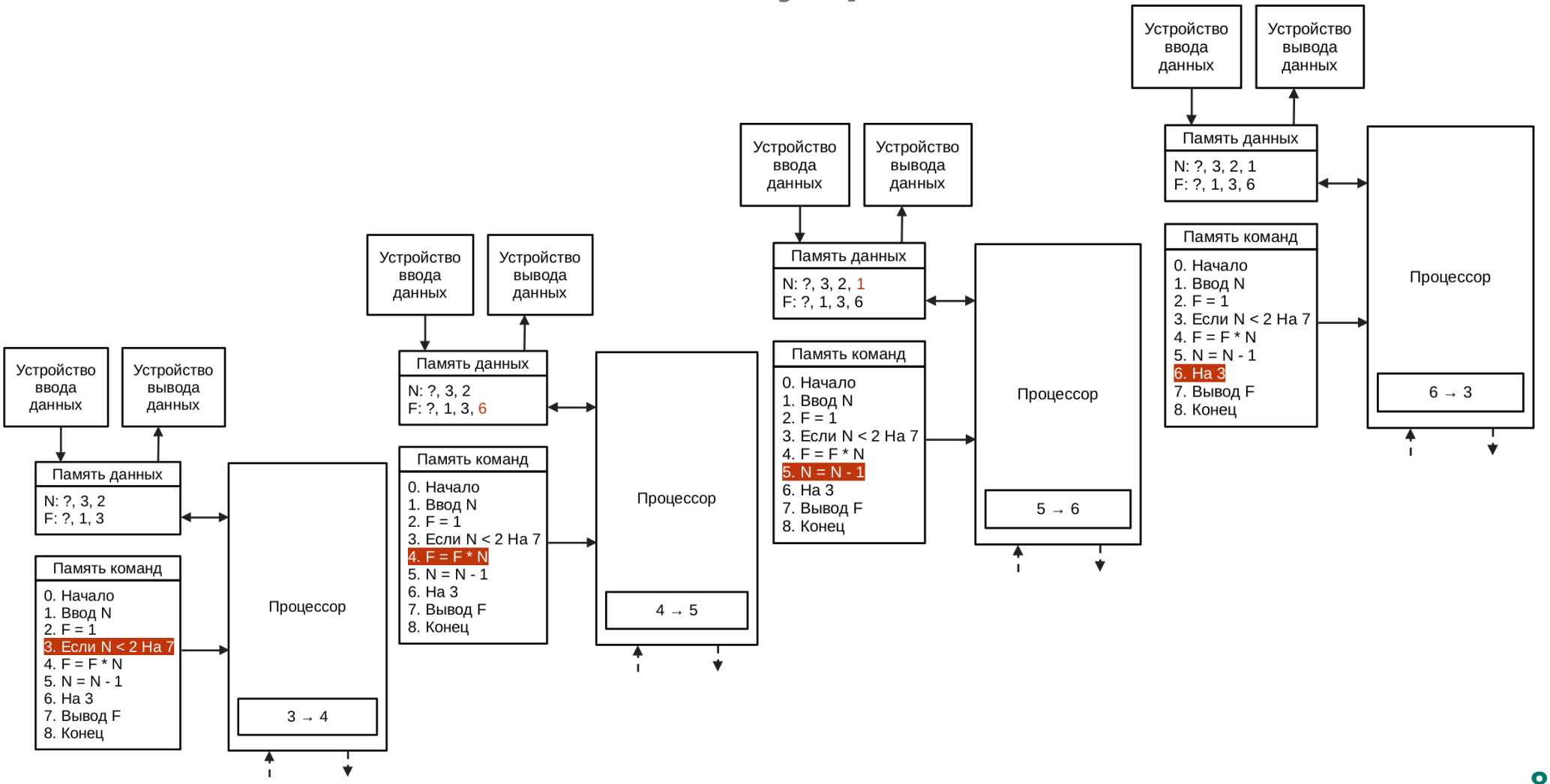

Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3
F: ?, 1, 3
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
4 → 5

Что внутри?

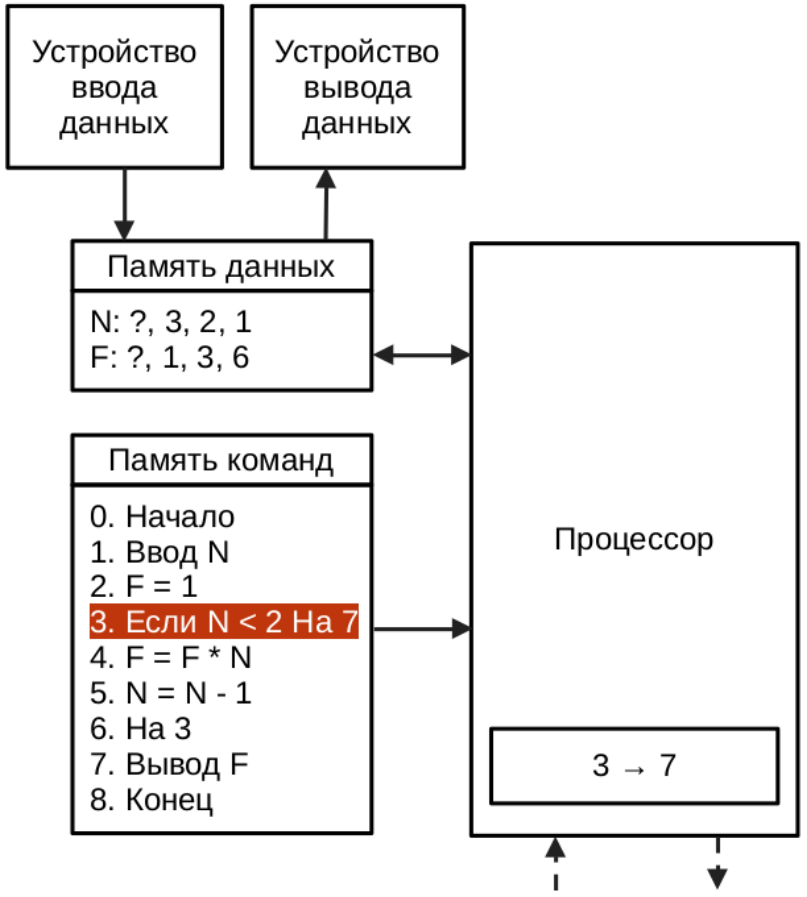

Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3, 2
F: ?, 1, 3
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
5 → 6

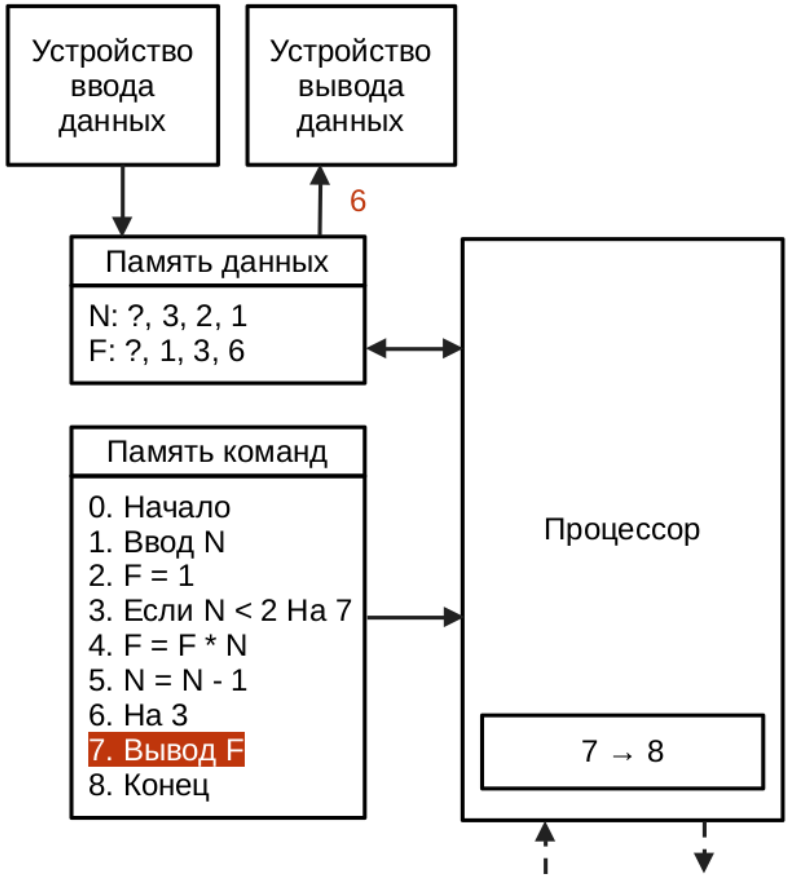

Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3, 2
F: ?, 1, 3
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
6 → 3

Что внутри?

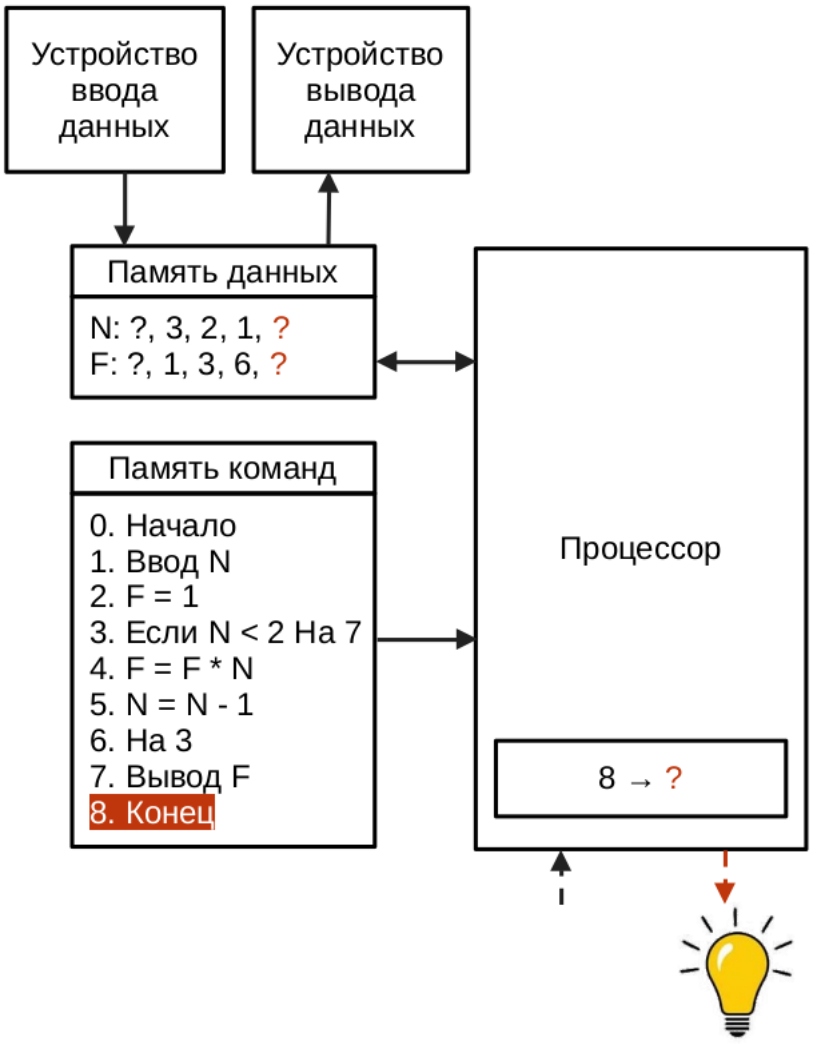

Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3, 2
F: ?, 1, 3
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
3 → 4
Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3, 2
F: ?, 1, 3, 6
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
4 → 5
Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3, 2, 1
F: ?, 1, 3, 6
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
5 → 6
Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3, 2, 1
F: ?, 1, 3, 6
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
6 → 3

Что внутри?


Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3, 2, 1
F: ?, 1, 3, 6
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
3 → 7


Устройство ввода данных
Устройство вывода данных
6
Память данных
N: ?, 3, 2, 1
F: ?, 1, 3, 6
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
7 → 8

Что внутри?


Устройство ввода данных
Устройство вывода данных
Память данных
N: ?, 3, 2, 1, ?
F: ?, 1, 3, 6, ?
Память команд
0. Начало
1. Ввод N
2. F = 1
3. Если N < 2 На 7
4. F = F * N
5. N = N - 1
6. На 3
7. Вывод F
8. Конец
Процессор
8 → ?

А компьютер выполнит этот алгоритм?

1) Можно ли использовать действительные числа, символы, строки?

2) Как воспринимается ввод данных?

3) Как отображаются данные?

4) На какие устройства ввода-вывода можно использовать?

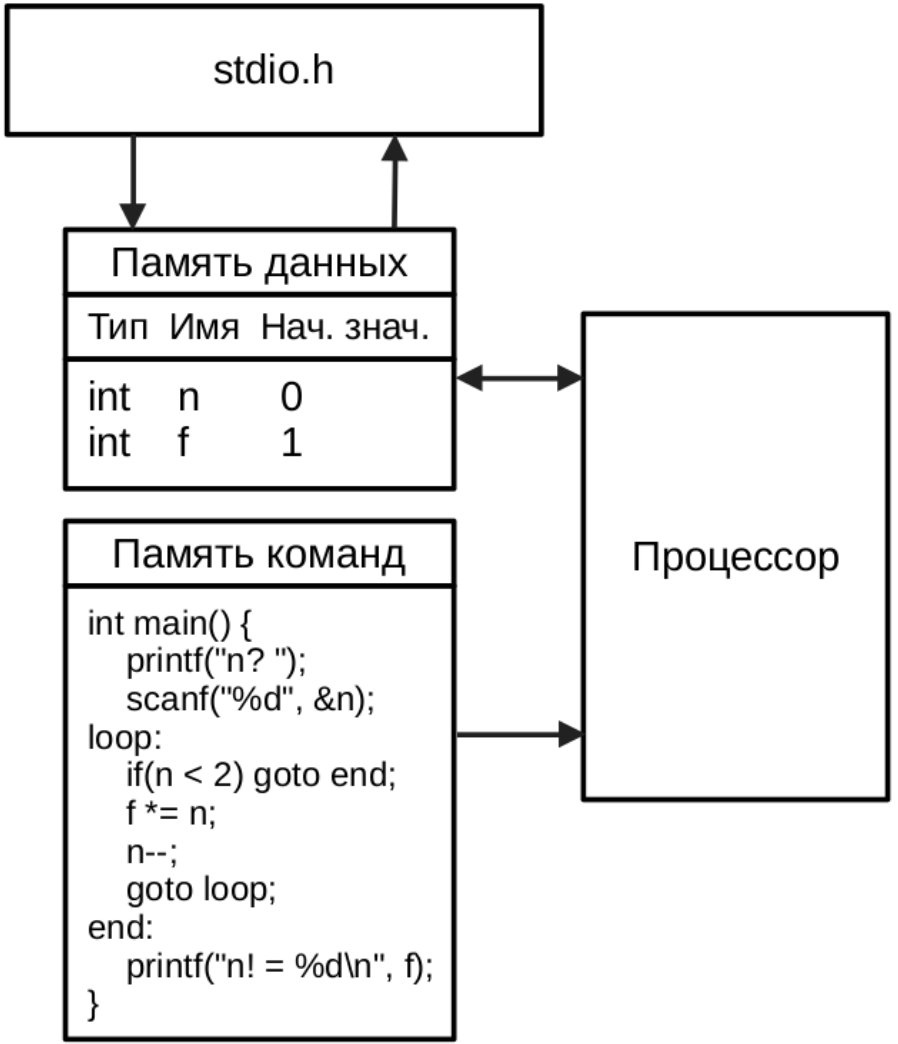
5) Какова семантика каждой выполняемой операции?

6) Какой тип памяти данных у N и F?

Программа для компьютера

```
1   #include <cstdio>
2
3   int n;
4   int f = 1;
5
6   int main() {
7       printf("n? ");
8       scanf("%d", &n);
9   loop:
10      if(n < 2) goto end;
11      f *= n;
12      n--;
13      goto loop;
14  end:
15      printf("n! = %d\n", f);
16  }
```

Отображение программы на структуру



Формирование однозначности операций

```
[ fact01]$ c++ fact.cpp
[ fact01]$ ./a.out
n? 5
n! = 120
```

```
1  #include <cstdio>
2
3  int n;
4  int f = 1;
5
6  int main() {
7                                        // Вовремя выполнения:
8      printf("n? ");                    // calc(char*)
9      scanf("%d", &n);                  // calc(char*); if(%d)-> use n as int
10 loop:
11                                       // Во время компиляции:
12     if(n < 2) goto end;               // <(int, int) -> bool
13     f *= n;                           // *(int, int) -> int; =(int) -> int
14     n--;                              // --(int) -> int
15     goto loop;
16 end:
17                                       // Вовремя выполнения:
18     printf("n! = %d\n", f);           // calc(char*); if(%d)-> use n as int
19 }
```

Формирование однозначности операций

```
[fact01]$ c++ bad_fact.cpp
[fact01]$ ./a.out
n? 5
n! = (null)
```

```
1  #include <cstdio>
2
3  int n;
4  int f = 1;
5
6  int main() {
7                                          // Вовремя выполнения:
8      printf("n? ");                      // calc(char*)
9      scanf("%c", &n);                    // calc(char*); if(%d)-> use n as int
10 loop:
11                                         // Во время компиляции:
12     if(n < 2) goto end;                 // <(int, int) -> bool
13     f *= n;                             // *(int, int) -> int; =(int) -> int
14     n--;                                // --(int) -> int
15     goto loop;
16 end:
17                                         // Вовремя выполнения:
18     printf("n! = %s\n", f);             // calc(char*); if(%d)-> use n as int
19 }
```

Способы задания однозначности

Однозначность определяет четкие правила выполнения операций реальными и виртуальными вычислительными системами, позволяя избегать или обходить ошибки программирования.

Различные методы задания однозначности операций позволяют контролировать корректность программы с разной степенью и на различных стадиях обработки

Выделяются:

1) Операционная однозначность (бестиповые системы)
2) Динамическая однозначность (системы с динамической типизацией)
3) Статическая однозначность (системы со статической типизацией)

Операционная однозначность

Однозначность операций формируется за счет четкого определения что и с какими типами данных делает каждая операция. Сами данные при этом не несут никакой дополнительной семантической идентификаии и представляются в виде набора строк бит (байт), размещенных в памяти. Доступ к обезличенным данным осуществляется по адресам, задаваемым в операциях. Для таких архитектур характерны бестиповые языки.

Примеры подобных архитектур:

1) Современные архитектуры уровня системы команд и их языки ассемблера
2) Объектно-ориентированный язык программирования Eolang
3) Языки системного программирования

Бестиповое программирование на C++

```
1  #include <cstdio>
2
3  char memory[2*sizeof(int)];      // Память для n и f
4  void* n = memory;                // Адрес на область для n
5  void* f = memory + sizeof(int); // Адрес на область для f
6
7  int main() {
8      *((int*)f) = 1;
9      printf("n? ");
10     scanf("%d", n);
11     printf("n = %d\n", *((int*)n));
12     printf("f = %d\n", *((int*)f));
13 loop:
14     if(*((int*)n) < 2) goto end;
15     *((int*)f) *= *((int*)n);
16     (*(int*)n)--;
17     goto loop;
18 end:
19     printf("n! = %d\n", (*(int*)f));
20 }
```

Динамическая однозначность

Динамическая однозначность операций формируется за счет того, что с каждым значением, формируемым в программе сопоставляется его тип. Любая операция над данным может проверить этот тип и выбрать в соответствии с этим нужные вычисления. То есть, одна и та же операция может обрабатывать различные типы данных. При этом идентификация типа осуществляется во время выполнения программы. Одни и те же переменные могут хранить данные различного типа. В любой момент программа может проверить тип переменной. Данный подход широко используется в языках программирования, ориентированных на интерпретацию.

Примеры:

1) Языки сценариев: Python, JavaScript, Lua…
2) Языки функционального программирования: Lisp...

Python. Использоание REPL (Read-Execute-Print Loop) для демонстрации изменения типа переменной

```
>>> value = 10
>>> value
10
>>> type(value)
<class 'int'>
>>> value = 3.14
>>> value
3.14
>>> type(value)
<class 'float'>
>>> value = "Hello!"
>>> value
'Hello!'
>>> type(value)
<class 'str'>
```

Python. Изменение и проверка типа в программе

```python
import random

for i in range(10):
    key = random.randint(1,2)
    if key == 1:
        value = random.uniform(1.0, 10.0)
    else:
        value = random.randint(100, 200)

    print('key = {0};  value = {1};  type = {2}'.format(key, value, type(value)))
```

```
key = 2;  value = 155;  type = <class 'int'>
key = 2;  value = 130;  type = <class 'int'>
key = 1;  value = 6.131331242406195;  type = <class 'float'>
key = 1;  value = 8.280520967840578;  type = <class 'float'>
key = 1;  value = 5.030964057739875;  type = <class 'float'>
key = 2;  value = 134;  type = <class 'int'>
key = 1;  value = 6.393939330816693;  type = <class 'float'>
key = 2;  value = 101;  type = <class 'int'>
key = 1;  value = 7.203902995902304;  type = <class 'float'>
key = 2;  value = 155;  type = <class 'int'>
```

Статическая однозначность

Статическая однозначность операций формируется за счет того, что с каждым значением в программе сопоставляется его тип. Этот тип задается при описании переменных и может быть проверен во время компиляции. Для всех временных и промежуточных значений тип может быть также выведен во время компиляции. Поэтому его не имеет смысла проверять во время выполнения. Одна и та же операция может быть задана с разными типами, но все вопросы по ее конкретному выполнению решаются во время компиляции (статический полиморфизм). С каждой переменной сопоставляется только один тип. Допускает эффективную трансформацию в бестиповые архитектуры уровня системы команд. Используется в языках компилируемого типа.

Примеры:

1) Императивные языки программирования:C, C++, Pascal, Oberon family, Java, C#, Rust, Go...

2) Языки функционального программирования: ML, Haskell...

Список источников информации по данной теме

1. [Википедия] Динамическая типизация
   https://ru.wikipedia.org/wiki/Динамическая_типизация
2. [Википедия] Статическая типизация
   https://ru.wikipedia.org/wiki/Статическая_типизация
3. Статическая и динамическая типизация
   https://habr.com/ru/post/308484/

Вопросы для обсуждения на семинаре

1. Основная идея однозначности выполнения операций. Способы достижения однозначности.
2. Достоинства и недостатки операционной однозначности.
3. Достоинства и недостатки динамической однозначности.
4. Достоинства и недостатки статической однозначности.
5. Связь между однозначность и методами типизации
6. Нужна ли динамическая проверка типов данных в статически типизированных языках?
7. Для чего в статически типизированных языках могут применяться бестиповые решения?
8. Когда в статически типизированных языках появляется необходимость динамической проверки типов?