Средства развития в ОИ языках - подпрограммы

Передача данных в подпрограммах

Связывание фактических и формальных параметров.

Способы передачи параметров:

1. по значению
2. по результату
3. по значению/результату
4. по адресу (ссылке)
5. по имени

Средства развития в ОИ языках - подпрограммы

Способы передачи параметров:

1. по значению
2. по результату
3. по значению/результату

Все три подразумевают отождествление формальных параметров и локальных переменных (то есть форм.п. и есть локальные переменные) => формальные параметры размещаются в стеке (точнее - в записи активации, другой термин - фрейм - activation record - frame). Семантика связывания - копирование значений фактических параметров в формальные (в стек) и обратно (из стека).

Вопрос - когда и куда (направление копирования)

1. по значению - *перед вызовом* п/п - фактические => формальные (очень удобно использовать команду ЯА - PUSH op)
2. по результату - *после завершения вызова* п/п - формальные => фактические (очень удобно использовать команду ЯА - POP op)
3. по значению/результату - очевидная комбинация этих двух способов

Средства развития в ОИ языках - подпрограммы

Способы передачи параметров:

1. по значению
2. по результату
3. по значению/результату

Чем удобно?

Явное и очевидное соответствие inout-семантике с уточнением вопроса, когда именно изменяются значения фактических параметров (либо никогда - случай 1, либо после завершения вызова - случаи 2 и 3)

Чем неудобно?

Если размеры параметров достаточно велики, то при копировании возникают накладные расходы

=> большинство ЯП применяют очевидный способ оптимизации - передача по адресу (ссылке)

Средства развития в ОИ языках - подпрограммы

Способы передачи параметров:

4. передача по адресу (ссылке)

В большинстве ЯП - передача адреса формального параметра "по значению".

То есть указатель на фактический параметр копируется в формальный перед вызовом, а операции разыменования указателя (ссылки) производятся компилятором автоматически

Очевидное ограничение - фактический параметр - только тот, для которого определено понятие ссылки (C - left-value).

Достоинство - при большом размере параметра эффективнее, чем предыдущие, реализует out и inout параметры (нет копирования).

Недостаток - обратная сторона достоинства - при небольшом размере параметра неэффективно обращение (всегда через косвенную адресацию), в то время как затратами на копирование можно пренебречь.

Средства развития в ОИ языках - подпрограммы

Способы передачи параметров:

5. передача по имени

Фактически - передается подпрограмма вычисления актуального адреса фактического параметра + информация для него. Чаще всего - просто адрес (для простой переменной).

```
integer i; integer array a[1:10];
i := 1;
procedure P (x); name integer x;
begin
        i := i+3;
        x := -1;
end;
P(a[i]); -- какой элемент массива a получит значение -1?
```

при изменении i - адрес параметра x меняется.

Немного похоже на макроподстановку....

Средства развития в ОИ языках - подпрограммы

Способы передачи параметров:

5. передача по имени

Фактически - передается подпрограмма вычисления актуального адреса фактического параметра + информация для него. Чаще всего - просто адрес (для простой переменной).

procedure swap(x,y); // как написать swap корректно?

Никак!

swap(i, a[i]) или swap(a[i], i) - что-то не заработает...

Алгол 60, Симула 67 - два способа - по значению и по имени - главная причина неэффективных реализаций этих языков - именно передача параметров по имени

Средства развития в ОИ языках - подпрограммы

Пример: Фортран - вначале - ТОЛЬКО по ссылке (характерно для ранних ЯП - почему?).

Первая проблема - некоторая неэффективность (в каких случаях?).

Некоторые реализации (Фортран) проблему неэффективности решали явной спецификацией параметров-результатов (3 способ). Дополнительное свойство ранних ЯП - отсутствие контроля межмодульных связей.

Вопрос - какая главная проблема?

В каких случаях?

Средства развития в ОИ языках - подпрограммы

Пример: Фортран - вначале - ТОЛЬКО по ссылке (характерно для ранних ЯП - почему?).

Первая проблема - некоторая неэффективность (в каких случаях?).

Некоторые реализации (Фортран) проблему неэффективности решали явной спецификацией параметров-результатов (3 способ). Дополнительное свойство ранних ЯП - отсутствие контроля межмодульных связей.

Вопрос - какая главная проблема?

Проблема ненадежности.

Задачник- "Задачи-'ловушки' на Фортране"

Решается - только при установлении контроля ММС(в поздних версиях - 90-е гг).

Средства развития в ОИ языках - подпрограммы

Пример: Ада 83 - компилятор выбирает между способами 1,2,3,4.

Очевидно, что 1 и 4 (по адресу и по ссылке) - самые универсальные, но используя в ряде случаев 2 и 3 (результату и значению/результату) можно получить ускорение за счет исключения косвенной адресации для примитивных базисных типов данных.

В чем проблема? В несовместимости реализаций.

P(A,A);

Средства развития в ОИ языках - подпрограммы

Пример: Паскаль - по значению (параметры-значения), по ссылке (параметры-переменные).

Чем плохо? Нет контроля inout-семантики.

Пример: С - единственный способ - по значению. По ссылке - моделируется "вручную" с помощью понятия указателя. Все разыменования - явные. Контроль за inout-семантикой - с помощью модификатора const.

void f(int x); void ff(int * x); void fff (const int * x);

Пример: С++ - фактически 2 способа передачи параметров - по адресу и по ссылке (если тип формального - ссылка). Есть выбор. Есть модификаторы const как для ссылок, так и (как в С - для указателей)

void f(int& x); void f(const int& x); // сравните с С

Средства развития в ОИ языках - подпрограммы

Пример: референциальные языки.

C#, Java, ... + все динамические ЯП (Python, JavaScript, ...)

Первый класс ЯП - референциальные типы + типы-значения.

Передача ссылки по значению === передача объекта по ссылке

Пример - Java (единственный способ - по значению для любых типов, но для реф-типов превращается в передачу по ссылке)

Что можно?

Вызывать методы объекта (объект может менять состояние, а может и нет - mutable/immutable objects), присваивать ссылку на объект другой ссылке.

Что нельзя? Менять значение фактического параметра.

```
class X {.....}
void f(X a) {  .....  a = new X();  ....}
X x1 = new X();   f(x1); // x1 ссылается на ТОТ ЖЕ САМЫЙ объект, что и до вызова f(x1)
```

Средства развития в ОИ языках - подпрограммы

Пример: референциальные языки.

C#, Java, ... + все динамические ЯП (Python, JavaScript, ...)

Первый класс ЯП - референциальные типы + типы-значения.

Передача ссылки по значению === передача объекта по ссылке

Пример - Java (единственный способ - по значению для любых типов, но для реф-типов превращается в передачу по ссылке)

Что можно?

Вызывать методы объекта (объект может менять состояние, а может и нет - mutable/immutable objects), присваивать ссылку на объект другой ссылке.

Что нельзя?

Средства развития в ОИ языках - подпрограммы

Пример: референциальные языки.

C#, Java, ... + все динамические ЯП (Python, JavaScript, ...)

Первый класс ЯП - референциальные типы + типы-значения.

Передача ссылки по значению === передача объекта по ссылке

Пример - Java (единственный способ - по значению для любых типов, но для реф-типов превращается в передачу по ссылке)

Что можно?

Вызывать методы объекта (объект может менять состояние, а может и нет - mutable/immutable objects), присваивать ссылку на объект другой ссылке.

Что нельзя?

Менять значение фактического параметра (формальный - меняйте на здоровье!)

Средства развития в ОИ языках - подпрограммы

Пример: референциальные языки.

C#, Java, ... + все динамические ЯП (Python, JavaScript, ...)

Первый класс ЯП - референциальные типы + типы-значения.

Передача ссылки по значению === передача объекта по ссылке

Пример - Java (единственный способ - по значению для любых типов, но для реф-типов превращается в передачу по ссылке)

Что можно?

Вызывать методы объекта (объект может менять состояние, а может и нет - mutable/immutable objects), присваивать ссылку на объект другой ссылке.

Что нельзя? Менять значение фактического параметра.

```
class X {.....}
void f(X a) {  .....  a = new X();  ....}
X x1 = new X();   f(x1); // x1 ссылается на ТОТ ЖЕ САМЫЙ объект, что и до вызова f(x1)
```

То же самое и для типов-значений (Java-терминология - примитивные типы)

Нельзя на Java написать функцию - аналог i++ (или ++i):

```
int i = 0, j;
j = i++; // или j = iPlusPlus(i);
int iPlusPlus (int i) { int retval = i; ++i; return retval; }
```

Средства развития в ОИ языках - подпрограммы

Пример - Java (единственный способ - по значению для любых типов, но для реф-типов превращается в передачу по ссылке)

Что можно?

Вызывать методы объекта (объект может менять состояние, а может и нет - ***mutable/immutable objects***), присваивать ссылку на объект другой ссылке.

А что же такое - ***mutable/immutable objects?***

В языке прямой поддержки нет. Можно только данные объекта объявлять неизменяемыми

Средства развития в ОИ языках - подпрограммы

Пример - C# - два способа - по значению, включая референциальные типы (то есть как в Java)+ по ссылке (ref и out параметры)

Примеры для Java - подходят и для C#

Но:

```
int iplusplus (ref int i) {
    return i++;
}
void makeX(out X a) { a = new X(); }
```

В чем отличие? ref - реализует inout-семантику, out - чистую out-семантику.

ВНИМАНИЕ!!! Требуется указывать модификатор при ВЫЗОВЕ:

```
int i = -1; // что будет если опустить инициализацию?
int j = iplusplus (ref i); // i== 0 , j == -1
X x;
makeX(out x);
```

Средства развития в ОИ языках - подпрограммы

Пример - C++ - два способа - по значению для всех + ссылочные типы по ссылке

inout-семантика: void f(X& x);

in-семантика: void f(const X& x);

+ r-value - ссылки (начиная с C++11).

Средства развития в ОИ языках - подпрограммы

Динамические ЯП (Python, JavaScript) - референциальная семантика => похоже на Java => передача ссылок по значению

Python:

```
def f(x):
        x = -3
a = 0
f(a)
print(a) # 0
arr = [1,2,3]
def ff(x):
        x[0] = -1
ff(arr)
print(arr) # [-1,2,3]
```

Абсолютно аналогично в JavaScript

Средства развития в ОИ языках - подпрограммы

Синтаксис передачи параметров:

- ключевые параметры
- параметры по умолчанию
- необязательные параметры
- список параметров переменной (произвольной) длины

Ключевые параметры - при вызове указывается имя параметра. Чаще всего в форме:

proc (ParamName: ParamValue)

f(to: Moscow, from : St-Petersburg, via: Tula)

Чем удобно?

- можно ставить в любом порядке
- легко расширить семантику на параметры по умолчанию и необязательные параметры

Часто использовался в старых ЯП (почему?)

Из современных: VB, Python, Swift

Средства развития в ОИ языках - подпрограммы

Ключевые параметры

Swift - 2 "фишки"

- внешние и локальные имена формальных параметров -внешние для обращений, локальные - для обозначения параметра в коде функции (ExternName LocalName: Type). Эти имена могут различаться
- если все параметры имеют внешние имена, то ключевой способ вызова ОБЯЗАТЕЛЕН. В то же время можно потребовать использовать только позиционный способ вызова для праметра, если запретить его внешнее имя.

func logMessage(message message: String, withPrefix prefix: String, andSuffix suffix: String)

Если внешние и локальные имена совпадают, то можем указывать только локальное имя, но первый параметр по умолчанию не имеет внешнего имени - всегда надо указывать, если хотим первый параметр вызывать ключевым образом.

Можно звать ТОЛЬКО через имена:

logMessage(message: "Полундра!!!", withPrefix: "<!--", andSuffix: "-->")

Если же:

func logMessage(_ message: String, _ prefix: String, _ suffix: String)

То можно ТОЛЬКО так:

logMessage("Полундра!!!", "<!--", "-->")

Средства развития в ОИ языках - подпрограммы

Ключевые параметры

Python - есть только вторая "фишка"

- любой параметр можно связать ключевым вызовом. Можно потребовать, чтобы параметры связывались только ключевым вызовом.

```
def f(a,b,c):
    ........
f(1,2,3)
f(a=1,b=2, c=3)
f(b=2,a=1,c=3)
def ff(*, a,b,c):
    ......
ff(1,2,3) # ошибка времени выполнения (другой и быть не может)
ff(a=1,b=2, c=3) # OK
ff(b=2,a=1,c=3) # OK
```

Средства развития в ОИ языках - подпрограммы

Параметры по умолчанию

В объявлении задается умолчательное значение параметра. Если в вызове этот параметр отсутствует, то транслятор подставляет значение по умолчанию.

Таким образом, код подпрограммы исходит из того, что параметр по умолчанию есть всегда, просто при вызове происходит подстановка

C++, C#, Python, Swift, VB

Complex(double re = 0.0, double im = 0.0);

Отсутствующие параметры

В отличие от предыдущего случая параметр реально может отсутствовать => должны быть средства, позволяющие коду подпрограммы узнать, есть параметр или нет.

VB - специальная функция - IsOptional(ArgName)

Достаточно редко - обычно - параметры по умолчанию, либо список параметров переменной длины

Средства развития в ОИ языках - подпрограммы

Список параметров переменной (произвольной) длины

C, C++ - <stdarg.h>

va_list va;

va_start(va, argName);

Type arg = va_next(va, Type);

va_end(va);

Достоинство - "работает" везде

Недостаток - (не)надежность.

Средства развития в ОИ языках - подпрограммы

Список параметров переменной (произвольной) длины

Объектно-ориентированные ЯП с единым корневым типом Object (SmallTalk, Java, C#, Delphi, ...)

Динамические ЯП

Очень простой способ - переменный список параметров - это формальный параметр - массив из Object или другого заданного типа(или просто массив переменных в динамических ЯП)

Надо просто указать транслятору, что фактические параметры нужно собрать в массив и связать его с формальным параметром.

Где? - В объявлении, конечно....

Средства развития в ОИ языках - подпрограммы

Список параметров переменной (произвольной) длины

C# (в точности из VB)

```
string LogMessage(string s, params object [] args) {
   StringBuilder sb = new StringBuilder(s);
   sb.Append(":");
   foreach (object o in args)
            sb.Append(" "); sb.Append(o.ToString());
    return sb.ToString();
}
Console.WriteLine(LogMessage(">>>", A, 32, file));
Console.WriteLine(LogMessage(">>>", x, "File error"));
// по этому принципу работают многие станд. методы:
int h = 8, m = 45;
string s = String.Format("Start time = {0}:{1}", h, m);
```

Средства развития в ОИ языках - подпрограммы

Список параметров переменной (произвольной) длины

Вместо Object - любой тип:

```
void f(params int[] args) { ... }
f(); // OK
f(1,2); // OK
f(1, "line"); // ошибка компиляции
```

Средства развития в ОИ языках - подпрограммы

<u>Список параметров переменной (произвольной) длины</u>

Java (с 2005 года):

void f (T ... args) { ... args - массив из T}

// заметим: for (x: arr) - цикл-итератор

// и автоупаковка и автораспаковка - без этого нельзя!

Средства развития в ОИ языках - подпрограммы

<u>Список параметров переменной (произвольной) длины</u>

Динамические ЯП - практически "даром"

Примеры - JavaScript - объект arguments - "подобный массиву"

Python:

есть еще и ключевые параметры

Позиционные параметры - массив(то есть список)

А ключевые параметры - что?

Средства развития в ОИ языках - подпрограммы

<u>Список параметров переменной (произвольной) длины</u>

Динамические ЯП - практически "даром"

Примеры - JavaScript - объект arguments - "подобный массиву"

Python:

есть еще и ключевые параметры

Позиционные параметры - массив(то есть список)

А ключевые параметры - что?

Словари!

Средства развития в ОИ языках - подпрограммы

Список параметров переменной (произвольной) длины

def foo(a, *args): # могу вместо *args - *someOtherNonStandardName

.... args - список аргументов, начиная со второго

foo(1) # args - пустой

foo("abc", 1,2,3) # args - [1,2,3]

def bar (*args, **kwargs)

... args - список позиционных параметров, kwargs - СЛОВАРЬ именованных параметров

bar (a=1, 25, c = "line", 2, b = 0) # args = [25,2], kwargs = {"a":1, "b":0, "c": "line"}

Почему - *.

Операция "распаковки" в список

Средства развития в ОИ языках - подпрограммы

Список параметров переменной (произвольной) длины

Пример:

```
def product(*numbers, initial=1):
    total = initial
    for n in numbers:
        total *= n
    return total

product(2,3,4,5)
product(2,3,4,5, initial = 2)
```

Средства развития в ОИ языках - подпрограммы

Список параметров переменной (произвольной) длины

Пример:

```
def format_attributes(**attributes):
    """Return a string of comma-separated key-value pairs."""
    return ", ".join(
        f"{param}: {value}"
        for param, value in attributes.items()
    )
# f-строки, генератор списка в выражениях, спецкомментарии
format_attributes(name="Trey", website="http://treyhunner.com", color="purple")
```

А можно и с произвольным словарем! Имена параметров - ЛЮБЫЕ!

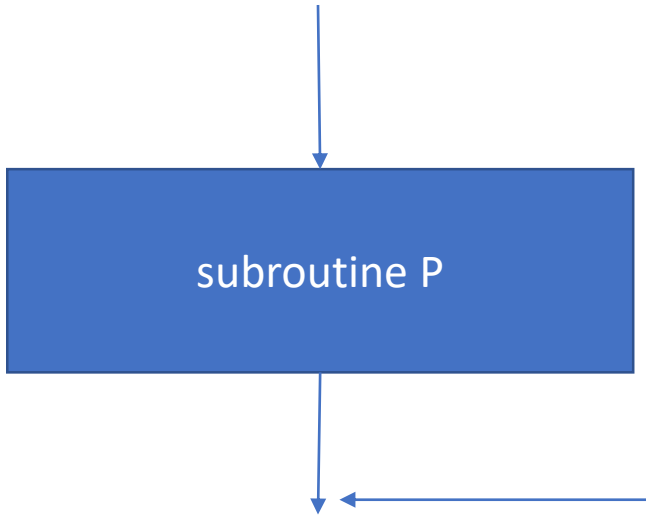
```
items = {'name': "Trey", 'website': "http://treyhunner.com", 'color': "purple"}
```

Средства развития в ОИ языках - подпрограммы

Передача управления в подпрограммах

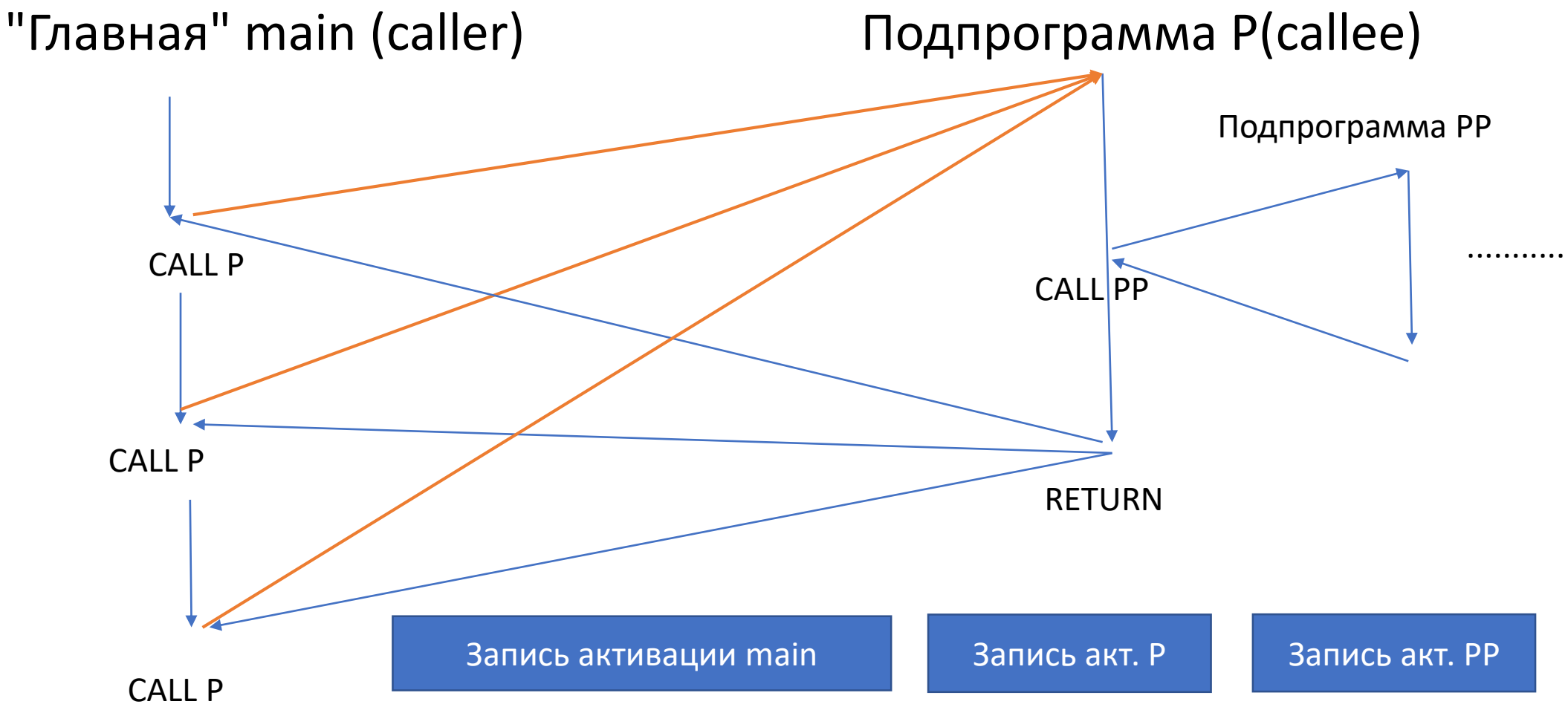
В современных ЯП - "черный ящик".

Раньше - ENTRY и т.п. Сейчас - нет.

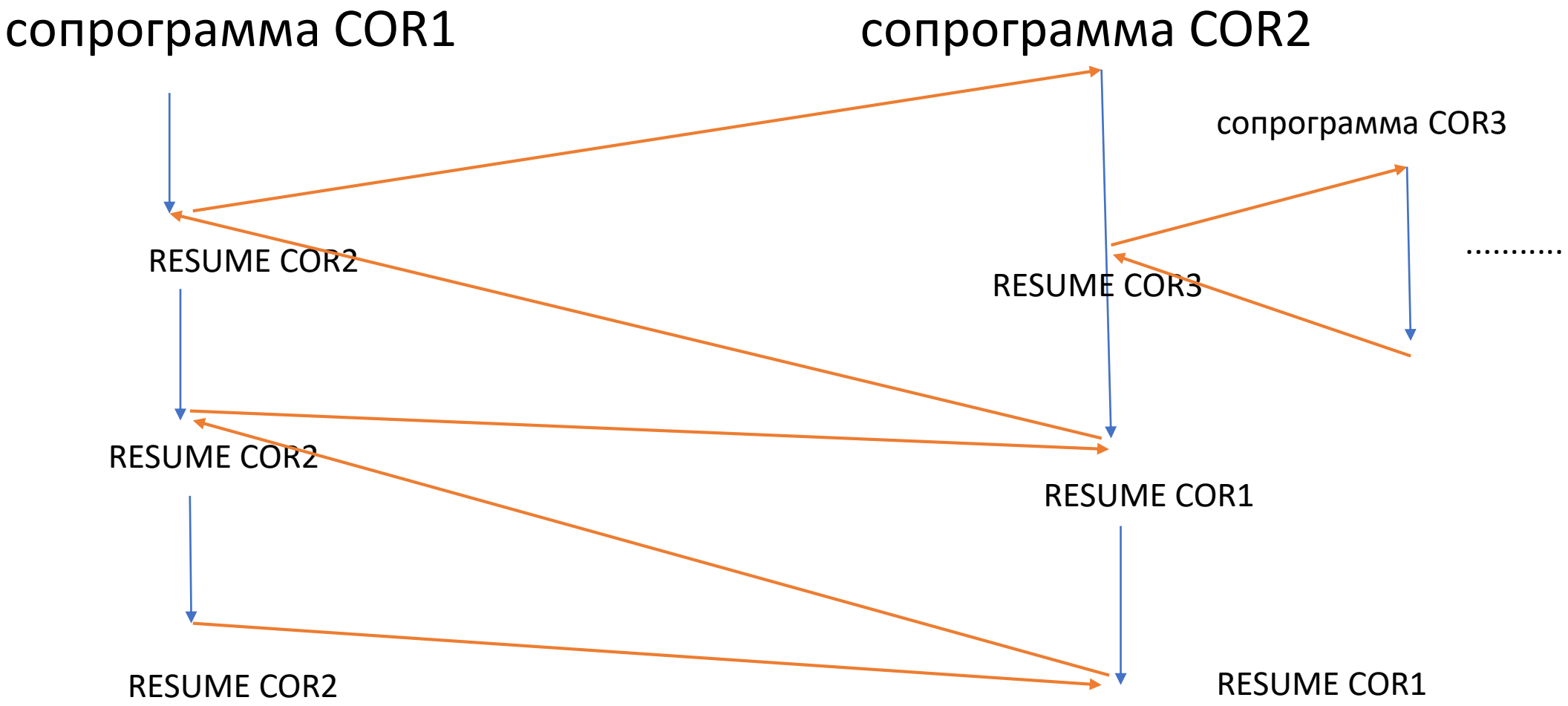


Внутри - как угодно (return и т.п.), но все собирается в один выход

Подпрограммы - поток управления



Сопрограммы - поток управления



Сопрограммы - поток управления

Концепция сопрограмм основана на сохранении модулем своего состояния между вызовами, включая сохранение точек, куда передается управление и откуда оно покидает модуль (замечание - не путать с множественными точками входа в подпрограммы).

Это отличает их от подпрограмм, которые не сохраняют информацию после отдачи управления (поэтому повторный вызов ВСЕГДА происходит с одной и той же начальной точки).

1958 год - Р. Конвей (опубликована статья в 1963 году). Реализации - Симула 67, Модула-2

Известны под именем "Сопрограммы Конвея-Кнута"

Сопрограммы - поток управления

Простой пример реализации - Модула 2

Всего 2 вызова: NEWPROCESS() и TRANSFER()

PROCEDURE NEWPROCESS(P:PROC; A: ADDRESS; N: CARDINAL; VAR Cor: ADDRESS);

PROCEDURE TRANSFER(VAR From, To: ADDRESS);

На основе сопрограмм реализован модуль Processes (квазипараллельные процессы).

Сопрограммы - поток управления

Простой пример реализации - Модула 2

```
DEFINITION MODELE Processes;
  TYPE SIGNAL;
  PROCEDURE StartProcess(P:PROC; N:CARDINAL);
  PROCEDURE Send(VAR S: SIGNAL);
  PROCEDURE Wait(VAR S: SIGNAL);
  PROCEDURE Awaited(S:SIGNAL):BOOLEAN;
  PROCEDURE Init(VAR S: SIGNAL);
END Processes;
```

Объем реализации (implementation module) - полторы страницы кода

+ концепция монитора

Сопрограммы - поток управления

Пример реализации сопрограмм Кнута на C (ANSI C!) - Simon Tatham

https://www.chiark.greenend.org.uk/~sgtatham/coroutines.html

80-90-ые годы - пропадание интереса к сопрограммам в новых языках программирования.

Гипотеза: вместо них в языки включались механизмы управления асинхронными параллельными вычислениями (явный параллелизм).

Реализации - задачи в языке Ада, JVM-потоки языка Java, потоки в .NET - C#.

Особняком стоит язык Erlang, концепция потоков которого предвосхитила соответствующие понятия в современных языках.

Сопрограммы - поток управления

Современные ЯП (2000-ые)

Возрождение интереса к концепции сопрограмм (без употребления такого термина)

Главное - сохранение состояния потока управления

Пример: язык C# - реализация итераторов

На прошлых лекциях - интерфейсы IEnumerable и IEnumerator

Сопрограммы - поток управления

Пример: язык C# - реализация итераторов

Простейший пример "старой" реализации

```
class Collection: IEnumerable {
        object [] arr = new object[3];
        public IEnumerator GetEnumerator() {
                return new Iterator(arr);
        }
}
```

Сопрограммы - поток управления

Пример: язык C# - реализация итераторов

Простейший пример "старой" реализации

```
class Iterator: IEnumerator
 {
		object []arr;
		int index;
		Iterator(object []a) {
			arr = a;
			index = -1;
		}
		bool MoveNext() {
			index++;
			return index < arr.length;
		}
		void Reset() { index = -1; }
		T Current {
			get { return arr[index]; }
		}
}
```

Сопрограммы - поток управления

Пример: язык C# - реализация итераторов

"Новая" (с 2002 года) реализация

```
class Collection: IEnumerable {
        object [] arr = new object[3];
        public IEnumerator GetEnumerator() {
                int index = 0;
                while (index < arr.length) {
                        yield return arr[index];
                        index++;
                }
                yield break;
        }
}
```

Сопрограммы - поток управления

Пример: язык C# - async - await - асинхронные потоки управления

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.IO;

namespace HelloApp
{
    class Program
    {
        static async void ReadWriteAsync()
        {
            string s = "Hello world! One step at a time";

            // hello.txt - файл, который будет записываться и считываться
            using (StreamWriter writer = new StreamWriter("hello.txt", false))
            {
                await writer.WriteLineAsync(s);  // асинхронная запись в файл
            }
            using (StreamReader reader = new StreamReader("hello.txt"))
            {
                string result = await reader.ReadToEndAsync();  // асинхронное чтение из файла
                Console.WriteLine(result);
            }
        }
        static void Main(string[] args)
        {
            ReadWriteAsync();

            Console.WriteLine("Некоторая работа");
            Console.Read();
        }
    }
}
```