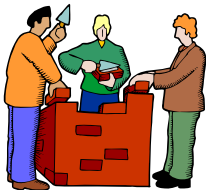
ПОДПРОГРАММЫ



Цивилизация развивается за счет расширения числа важных операций, которые можно выполнять, не думая о них.

А.Н. Уайтхед

Языки программирования

Содержание

2

- Понятие подпрограммы
  - Прототип подпрограммы
  - Виды параметров подпрограмм
  - Виды локальных переменных подпрограмм
- Передача параметров в подпрограмму
  - Модели передачи параметров и их реализация
  - Многомерные массивы и подпрограммы как параметры подпрограмм
  - Проверка типов параметров
  - Перегрузка подпрограмм
  - Передача параметров в основных языках программирования
- Раздельная и независимая компиляция подпрограмм

Подпрограммы

3

- *Подпрограмма* – синтаксически обособленная часть программы, решающая отдельную подзадачу.
  - Подпрограмма не выполняется сама по себе: всегда *вызывается* прямо или косвенно из основной программы или из другой подпрограммы.
  - Оператор вызова подпрограммы заменяет набор операторов в подпрограмме.
- Общие свойства подпрограмм
  - Один вход.
  - На время выполнения подпрограммы выполнение вызывающего модуля откладывается.
  - После выполнения подпрограммы управление возвращается в вызывающий модуль.

Прототипы подпрограмм

4

```
void InpData(int *, float *, char *);
float Calc(int, float, char, float *);
void OutData(float, float);

int a;
float b;
char c;
float res1, res2;

void main(void)
{
  InpData(&a, &b, &c);
  res1=Calc(a, b, c, &res2);
  OutData(res1, res2);
}
```

```
void InpData(int *, float *, char *)
{ … }

float Calc(int, float, char, float *)
{ … }

void OutData(float, float)
{ … }
```


Основные определения

5

- Заголовок (прототип) подпрограммы:
  - ключевое слово, определяющее начало описания подпрограммы
  - имя подпрограммы
  - список параметров подпрограммы
- *Параметры* – альтернатива прямого доступа подпрограммы к глобальным переменным.
  - *Формальные* параметры – фиктивные переменные, указываются в заголовке подпрограммы.
  - *Фактические* параметры – указываются при вызове подпрограммы.
- Виды подпрограмм
  - *Процедура* – подпрограмма, не возвращающая значение.
  - *Функция* – подпрограмма, возвращающая значение (результат).
    - C, C++: процедуры – функции, возвращающие результат типа void.

Параметры

6

- Позиционные
  - Pascal
    - MyProc(p1, p2, p3, p4);
- Ключевые
  - Ada
    - MyProc(p1, Param3=>p3, Param2=>p2);
- По умолчанию
  - Ada
    - function MyFunc(p1: Float; p2: Integer:=1; p3: Float) return Float;
    - res:=MyFunc(2.0, p3=>1.5);
  - C++
    - float MyFunc(float p1, float p2, int p3=1);
    - res=MyFunc(2.0, 1.5);

Локальные переменные подпрограмм

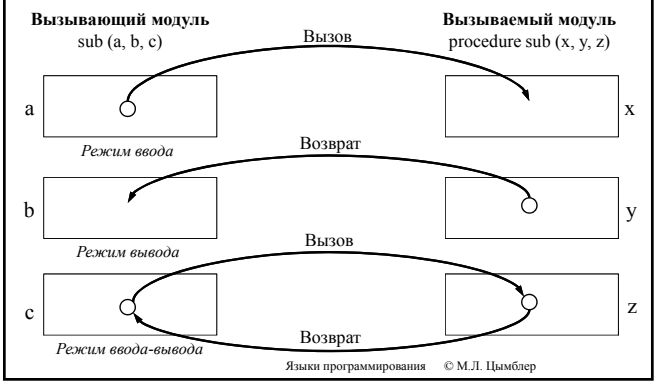
7

- *Автоматические* локальные переменные помещаются в сегмент стека в начале выполнения подпрограммы и удаляются оттуда по окончании ее выполнения.
  - Pascal, Modula-2, Ada: только автоматические локальные переменные.
- *Статические* локальные переменные помещаются в сегмент стека в начале выполнения подпрограммы и остаются там до окончания выполнения программы.
  - C, C++

```
int MyFunc(int p1, float p2)
{
  static int callcnt=0;
  …
}
```


Модели передачи параметров

8



Модели реализации передачи параметров

9

- Передача по значению (pass-by-value)
- Передача по результату (pass-by-result)
- Передача по значению и результату (pass-by-value-result)
- Передача по ссылке (pass-by-reference)
- Передача по имени (pass-by-name)

Передача по значению

10

- Семантика
  - Реализует семантику режима ввода. Значение фактического параметра используется для инициализации соответствующего формального параметра, который в дальнейшем действует как локальная переменная.
- Реализация
  - Обычно путем реальной передачи данных.
  - Может быть реализована с помощью передачи пути доступа к значению фактического параметра в вызывающем модуле. В этом случае необходимо обеспечить защиту от записи ячейки, хранящей это значение.
- Преимущества
  - Эффективность доступа к данным.
- Недостатки
  - Требует дополнительной памяти для хранения формального параметра (в вызываемой подпрограмме или вне ее и вызывающего модуля).
  - Операция перемещения может быть дорогостоящей (например, длинный массив).

Передача по результату

11

- Семантика.
  - Реализует семантику режима вывода. При передаче параметра никакое значение в подпрограмму не передается. Соответствующий формальный параметр действует как локальная переменная, но непосредственно перед возвращением управления обратно в вызывающий модуль его значение передается фактическому параметру, который должен представлять собой переменную.
- Реализация.
  - Обычно путем реальной передачи данных.
- Недостатки.
  - Может возникнуть коллизия фактических параметров: sub(p1, p1).
  - Низкая мобильность программ, т.к. разработчик средств реализации языка определяет, когда вычисляется адрес фактического параметра: во время вызова подпрограммы или при возвращении из нее.

Передача по значению и результату

12

- Семантика.
  - Реализует семантику режима ввода-вывода, в котором фактические значения физически перемещаются.
- Реализация
  - Значение фактического параметра используется для инициализации соответствующего формального параметра, который затем действует как локальная переменная.
  - Формальные параметры должны храниться в локальной области памяти, связанной с вызываемой подпрограммой. При завершении выполнения подпрограммы значение формального параметра передается обратно фактическому параметру.
- Недостатки.
  - Необходимость хранить параметры в нескольких местах и тратить время на копирование их значений.
  - Может возникнуть коллизия фактических параметров.

Передача по ссылке

13

- Семантика.
  - Реализует семантику режима ввода-вывода. Передается путь доступа к данным (адрес вместо данных) в вызываемую подпрограмму.
- Преимущества.
  - Эффективность по времени и использованию памяти.
- Недостатки.
  - Более медленный доступ к формальным параметрам (необходим еще один уровень косвенной адресации при передаче данных).
  - Неумышленные и ошибочные изменения фактического параметра.

Передача по имени

14

- Семантика.
  - Не соответствует какой-либо модели. Фактический параметр буквально заменяется соответствующим формальным параметром во всех местах подпрограммы.
- Реализация.
  - Формальный параметр связывается с методом доступа во время вызова программы, однако фактическое связывание с некоторым значением или адресом откладывается, пока формальному параметру не будет присвоено какое-либо значение, либо на него не будет сделана ссылка.
  - Если фактический параметр – скаляр, то передача по имени равносильна передаче по ссылке; если константное выражение – равносильна передаче по значению.
  - Если фактический параметр – элемент массива, то передача по имени может отличаться от любого другого метода (значение индексного выражения может изменяться между моментами времени, когда производятся разные обращения к этому параметру).
- Преимущества.
  - Гибкость.
- Недостатки.
  - Медленно по сравнению с др. методами передачи параметров.
  - Трудно реализовать.
  - Некоторые простые операции невозможно выполнить при передаче параметров по имени (например, exchange(a, b)).

Передача по имени

15

- ALGOL
  - procedure P;
    Integer Global;
    Integer array List[1:2];
    procedure S (param);
      integer param;
    begin
      param:=3;
      Global:=Global+1;
      param:=5;
    end;
  begin
    List[1]:=2;
    List[2]:=2;
    Global:=1;
    S(List[Global]);
  end;

- После выполнения P массив List содержит значения 3 и 5, которые устанавливаются в S.
- Доступ к List[2] открывается, когда значение Global увеличивается на 1 в S и становится равным 2.

Передача параметров в основных языках программирования

16

- FORTRAN
  - Используется модель ввода-вывода, но не указывается способ передачи (по ссылке или по значению и результату). До FORTRAN 77 – по ссылке, после – по значению и результату.
- ALGOL 60
  - Впервые введена передачи по имени. Возможна также передача по значению.
- C
  - Передача по значению. Семантика передачи по имени реализуется с помощью применения указателей в качестве параметров.
- C++
  - Параметры-ссылки реализуют передачу по ссылке. Разыменовываются неявно.
    - void func(const int **&**p1, int p2, int **&**p3);

Передача параметров в основных языках программирования

17

- Java
  - Передача по значению. Однако доступ к программным объектам в Java возможен только с помощью ссылок, поэтому параметры в действительности являются ссылками.
- ALGOL W
  - Впервые введена передачи по значению и результату.
- Pascal, Modula-2
  - Передача по значению (по умолчанию) и передача по ссылке (var).
- Ada
  - out – можно присваивать любые значения, но нельзя ссылаться;
  - in – можно ссылаться, но нельзя ничего присваивать;
  - in out – можно присваивать значения и ссылаться.

Проверка типов параметров

18

- FORTRAN 77
  - Не нужна.
- Pascal, Modula-2, FORTRAN 90, Java, Ada
  - Не предусмотрена.
- C
  - int func(p1, p2, p3)
double p1;
int p2;
char * p3;
{ … }
  - Если приведение типа фактического параметра к типу формального параметра невозможно или не верно количество параметров, то синтаксическая ошибка.
- C++
  - Проверки типов некоторых параметров можно избежать:
    - printf(const char * …); // % указывает на наличие других параметров

Многомерные массивы как параметры подпрограмм

19

- C
  - #define M 100
    void func(int matrix[][M]) { … }
    int m[10][10]
    void main()
    {
      func(m);
    }
  - #define M 100
    void func(int *matrix, int num_rows, int_num_cols)
    {
    #define matrix(i,j) (*(matrix+((i)*M)+(j)))
      matrix(i,j)=x;
    …
    }

Примеры

20

Pascal	procedure Exchange(a,b: Integer);
	var tmp: Integer;
	begin
	  tmp:=a;
	  a:=b;
	  b:=tmp;
	end;
		procedure Exchange(var a,b: Integer);
		var tmp: Integer;
		begin
		  tmp:=a;
		  a:=b;
		  b:=tmp;
		end;

C	void Exchange(int a,b);
	{
	  int tmp;
	
	  tmp=a;
	  a=b;
	  b=tmp;
	}
		void Exchange(int *a,*b);
		{
		  int tmp;
		
		  tmp=*a;
		  *a=*b;
		  *b=tmp;
		}

Примеры

21

C++	void Exchange(int a,b);
	{
	  int tmp;
	
	  tmp=a;
	  a=b;
	  b=tmp;
	}
		void Exchange(int &a,&b);
		{
		  int tmp;
		
		  tmp=a;
		  a=b;
		  b=tmp;
		}

Ada		procedure Exchange(a in out Integer,
		b in out Integer) is
		tmp: Integer;
		begin
		  tmp:=a;
		  a:=b;
		  b:=tmp;
		end Exchange;

Процедуры как параметры подпрограмм

22

- Pascal
  - type
    Func=function (x: Real): Real;
    function Integrate(f: Func; a, b, Eps: Real): Real;
- C
  - typedef float (* func_t)(float);
    float Integrate(func_t f, float a, float b, float Eps);

Перегруженные подпрограммы

23

- *Перегруженная подпрограмма* – это подпрограмма, имя которой совпадает с именем другой подпрограммы в той же среде ссылок, но отличается количеством / порядком / типами параметров / типом возвращаемого результата.
  - Ada
    procedure MAIN is
    type FLOAT_VECTOR is array (INTEGER range <>) of FLOAT;
    type INT_VECTOR is array (INTEGER range <>) of INTEGER;
    procedure Sort(A : in out FLOAT_VECTOR) is …
    end Sort;
    procedure Sort(A : in out INT_VECTOR) is …
    end Sort;
    end MAIN;

Настраиваемые подпрограммы

24

- Ada
  - generic
    type INDEX_TYPE is (<>);
    type ELEMENT_TYPE is private;
    type VECTOR is array (INTEGER_TYPE range <>) of ELEMENT_TYPE;
    procedure GENERIC_SORT(LIST : in out VECTOR) is …
    end GENERIC_SORT;
  - procedure INT_SORT is new
    GENERIC_SORT(INDEX_TYPE=>INTEGER,
    ELEMENT_TYPE=>INTEGER,
    VECTOR=>INT_ARRAY);

Перегружаемые операторы как подпрограммы

25

- *Перегруженный оператор* – это оператор, значение которого определяется типами его операндов.
  - C
    - int * int, float * float
  - Ada
    - function "*" (A, B : in Vector) return INTEGER is
      SUM: INTEGER:=0;
      begin
        for INDEX in A'range loop
          SUM:=SUM+A(INDEX)*B(INDEX);
        end loop;
        return SUM;
      end "*";
  - C++
    - int operator * (const vector &a, const vector &b, int len);

Контрактное программирование

26

- Eiffel
  - routine Divide(dividend, divisor, result, remainder: integer)
    is
    require
      divisor>0
    do
      …
    ensure
      dividend=result*divisor+remainder;
      remainder<divisor;
    end;

Раздельная и независимая компиляция подпрограмм

27

- *Раздельная компиляция* программных объектов означает, что единицы компиляции могут компилироваться в разное время, но при этом процессы их компиляции зависят друг от друга, если ими используются сущности других (внешних) единиц компиляции.
  - Допускают языки Ada, Modula-2, FORTRAN 90.
- *Независимая компиляция* допускает компиляцию программных объектов без какой-либо информации о сущностях внешних единиц компиляции.
  - Допускают языки C, FORTRAN 77.
- Язык Pascal не допускает ни раздельной, ни независимой компиляции.