Chương 4: Chương trình con (SubPrograms)

Giảng viên: Ph.D Nguyễn Văn Hòa

Khoa KT-CN-MT – ĐH An Giang

Pascal Code Fragment

```
procedure C;
  procedure A (P : procedure; i : integer);
      procedure B;
      begin B
           write(i);
       end B;
  begin A
      if i = 1 then A(B,2)
      else P;
  end A;
begin main
  A(C,1);
end main.
```

JavaScript Code Fragment

```
function sub1() {
    var x;
    function sub2() {
        alert(x);
    };
    function sub3() {
        var x;
        x = 3;
         sub4(sub2);
    };
    function sub4(subx) {
        var x;
        x = 4;
        subx();
     };
    x = 1;
    sub3();
};
```

Chương trình con chung C++

```
template <class Type>
   Type max(Type first, Type second) {
       return first > second ? first : second;
   }
int max(int first, int second)
{
   return first > second ? first : second;
}
```

Nội dung chính của chương

- Giới thiệu chương trình con
- Cơ chế gọi chương trình con
- Truyền tham số cho chương trình con
- Chương trình con đa năng (overloaded)
- Chương trình con chung (generic)

Giới thiệu

- Có hai cách trù tượng hóa
  - Trù tượng tiến trình (process abstraction): được chú trọng ngày từ rất sớm
  - Trù tượng dữ liệu (data abstraction): được chú trọng trong 1980s
- Chương trình con (CTC):
  - Một phép toán trừu tượng tiến trình (process) được định nghĩa bởi người lập trình
  - Khi một khối công việc được lặp đi lặp lại nhiều lần trong chương trình → CTC
  - Hoặc CTC được dùng để tách một khối công việc cụ thể, để chương trình chính đỡ phức tạp

Giới thiệu (tt)

- Đặc tính cơ bản của CTC
  - Mỗi chương trình con có một điểm vào duy nhất
  - Chương trình gọi CTC thì tạm dừng trong khoảng thời gian thực hiện CTC
  - Điều khiển luôn được trả về chương trình gọi khi kết thúc chương trình con
  - Mô hình Master/Client
- Hai khía cạnh khi nói đến CTC
  - Định nghĩa CTC
  - Lời gọi CTC

Giới thiệu (tt)

- CTC có thể truy xuất dữ liệu :
  - Truy xuất các biến không cục bộ
  - Truyền tham số
- Ưu điểm của CTC
  - Cho phép sử dụng nhiều lần 1 chức năng/khối công việc ~ CTC → tiết kiệm không gian lưu trữ code và ẩn giấu các chi tiết của CT
  - Tăng tính dễ đọc hiểu của CT vì dễ dàng thấy cấu trúc điều khiển của CT hơn
  - Phát hiện và sữa lỗi dễ dàng

Mô hình cài đặt CTC

- Mô hình cài đặt của CTC trong các NNLT có thể khác nhau
  - Điều khiển tuần tự (Imperative) :
    - Thủ tục : một khối các câu lệnh để thực hiện 1 chức năng
    - Hàm : một khối câu lệnh trả về 1 kết quả duy nhất
    - Ngôn ngữ C không phân biệt hàm và thủ tục
  - Hàm: VD Hàm tính dãy Fibonacci
  - Logic: Mệnh đề Horn (Horn claus)

Đặc tả của CTC

- Tên của CTC
- Số lượng, thứ tự và kiểu của các tham số (đối số)
  - Tham số hình thức: là danh sách các tham số được dùng trong CTC ở phần Header của CTC
  - Tham số thực: là các giá trị hoặc địa chỉ ô nhớ được dùng trong lời gọi CTC
  - Header CTC = Tên + tham số hình thức
- Hoạt động của CTC hay phần thân (body)
  - Các khối như các khai báo, các câu lệnh, etc
- Số lượng kết quả trả về và kiểu của chúng

VD CTC (thủ tục) trong Pascal

```
procedure count(k: array[1..5] of real);
    const
        <constant-declarations>
    type
        <type-declarations>
    var
        <variable-declarations>
    // nested procedures and functions go here
    begin
        <statements>
    end;
```

VD CTC (thủ tục) trong Ada

```
procedure Display_Even_Numbers is
    < declarations>
    function even (number:integer) return boolean is
      begin
      <statements>
    end even;
    begin
    <statements>
    end Display_Even_Numbers;
```

Các yếu tố khi thiết kế CTC

- Các hình thức truyền tham số: tham trị hay quy chiếu,…?
- Có kiểm tra kiểu hay không?
- Các biến cục bộ là tĩnh (static) hay động?
- Một CTC có thể được khai báo lòng vào một CTC khác không?
- CTC có được đa năng hóa (overloaded) không?
- CTC là chung hay không (generic subprogram)?

Các biến cục bộ (local) của CTC

- Các biến cục bộ động stack
  - Liên kết vào các ô nhớ khi CTC bắt đầu được thực hiện và hủy liên kết khi kết thúc CTC
  - Ưu điểm
    - Hỗ trợ đệ qui
    - Ô nhớ dành cho các biến cục bộ có thể được shared giữa các CTC
  - Khuyết điểm
    - Cần thời gian cấp, giải phóng và khởi tạo
    - Không thể lưu giá trị của biến giữa các lần gọi CTC
- Các biến cục bộ tĩnh
  - Hiệu quả hơn
  - Không hỗ đệ qui
  - Không thể chia sẽ các ô nhớ

Các biến cục bộ của CTC (tt)

- Trong C và C++ biến cục bộ được khai báo tĩnh nếu đứng sau *static*

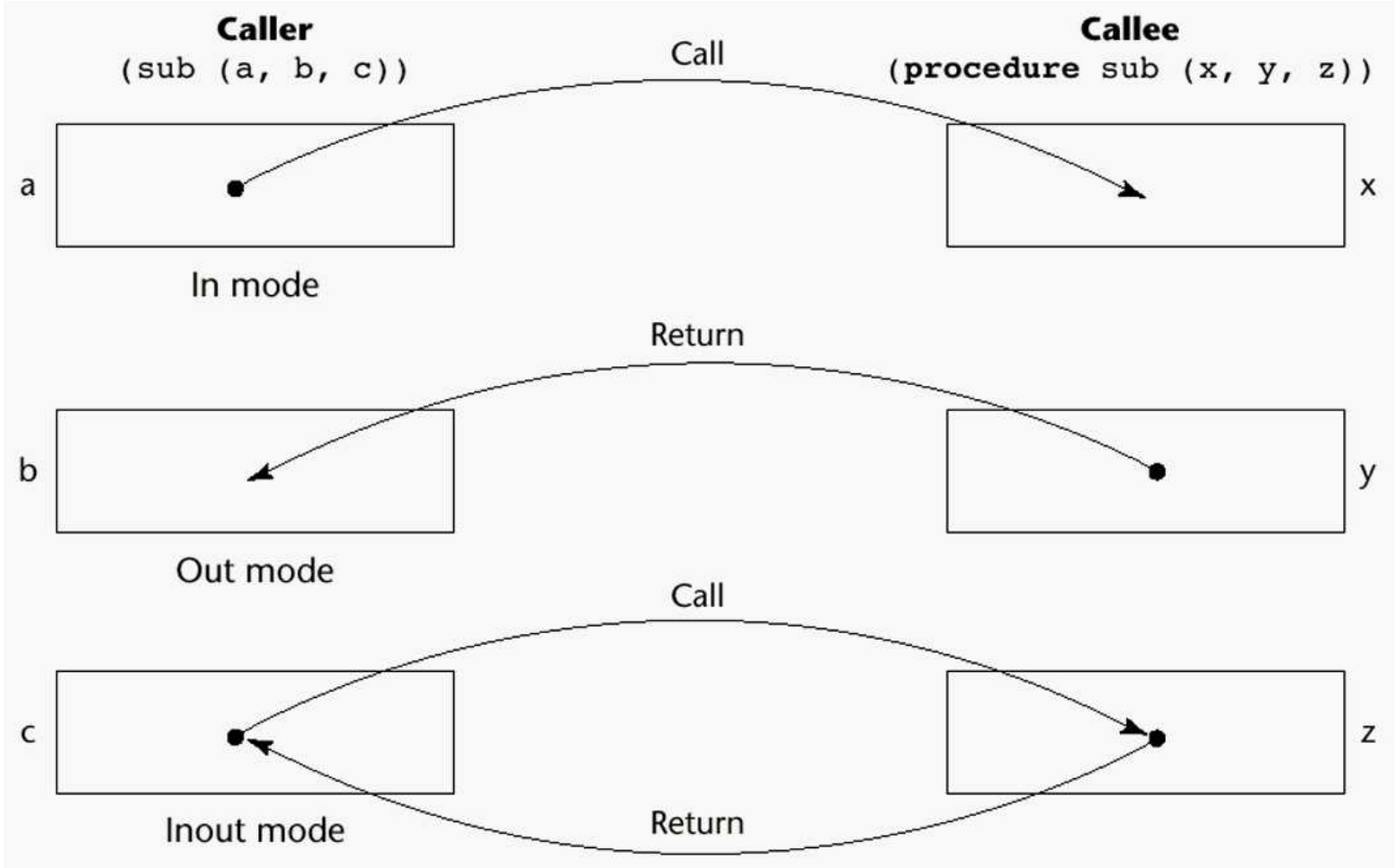
```
int adder (int list[], int listlen){
    static int sum = 0;
    int count;
    for (count=0; count<listlen;count++)
    sum += list[count];
    return sum;
}
```

- // count là biến cục bộ động stack
- // sum biến tĩnh

Truyền tham số

- Khi gọi CTC, các tham số được truyền bằng một trong các cách sau đây :
  - Truyền tham trị (Pass-by-value)
  - Truyền kết quả (Pass-by-result)
  - Truyền trị và kết quả (Pass-by-value-result)
  - Truyền quy chiếu (Pass-by-reference)

Các mô hình truyền tham số



Truyền tham trị - In Mode

- Giá trị của tham số thực được dùng để truyền vào tham số hình thức tương ứng
  - Cách cài đặt bình thường là copy
  - Có thể cài đặt bằng cách truyền địa chỉ nhưng cách này không được khuyến khích (vì đòi hỏi biến phải được đặt ở chế độ write-protection)
  - Khi tác vụ copy được dùng → cần thêm không gian lưu trữ
  - Lưu trữ và tác vụ copy có thể mất thời gian
- Trị cuối cùng của tham số thực bị mất khi CTC kết thúc

Truyền tham trị - In Mode (tt)

- Các NNLT hỗ trợ : C, Pascal, Ada, Scheme, Algol68

```
{ c : array [1..10] of integer;
  m,n: integer;
  procedure r(k,j: integer);
    begin
       k:=k+1; /* m = 6 */
       j:= j+2;  /* n = 5 */
    end;
  begin
  m := 5; n:=3;
  r(m,n);
  writeln(m,n); /* 5 & 3 */
}
```

Truyền kết quả - Out Mode

- Tham số thực không truyền giá đến CTC; tham số hình thức tương ứng đống vai trò như biến cục bộ nhưng khi kết thúc CTC thì trị của tham số này được trả về cho tham số thực
  - Yêu cầu không gian lưu trữ và tác vụ copy
  - Tham số thực phải là 1 biến
- Khả năng bị đụng độ về tham số
  - `Sub(p1,p1);` một khi tham số hình thức được copy trở lại thì lần copy sau cũng thể hiện trị của p1
- NNLT hỗ trợ : Ada

Truyền tham trị & kết quả - Inout Mode

- Sự kết hợp truyền trị và truyền kết quả (pass-by-value and pass-by-result)
- Tham số hình thức cần không gian lưu trữ cục bộ
- Tham số hình thức phải là 1 biến (có ô nhớ), copy trị
- Giá trị cuối cùng của tham số hình thức được copy cho tham số thực
- Khuyết điểm:
  - Các khuyết điểm của truyền tham trị
  - Các khuyết điểm của truyền kết quả
- NNLT hỗ trợ : Fortran

Truyền tham trị & kết quả (tt)

```
{ c : array [1..10] of integer;
  m,n: integer;
  procedure r(k,j: integer);
     begin
       k:=k+1;
       j:= j+2;
     end;
  begin
  /* set c[m] = m*
  m := 2;
  r(m,c[m]);
  write(c[1],c[2],..,c[10]);  /* Gía trị của c[2] hay c[3] bị thay đổi */
}
```

Truyền quy chiếu - Pass by Reference

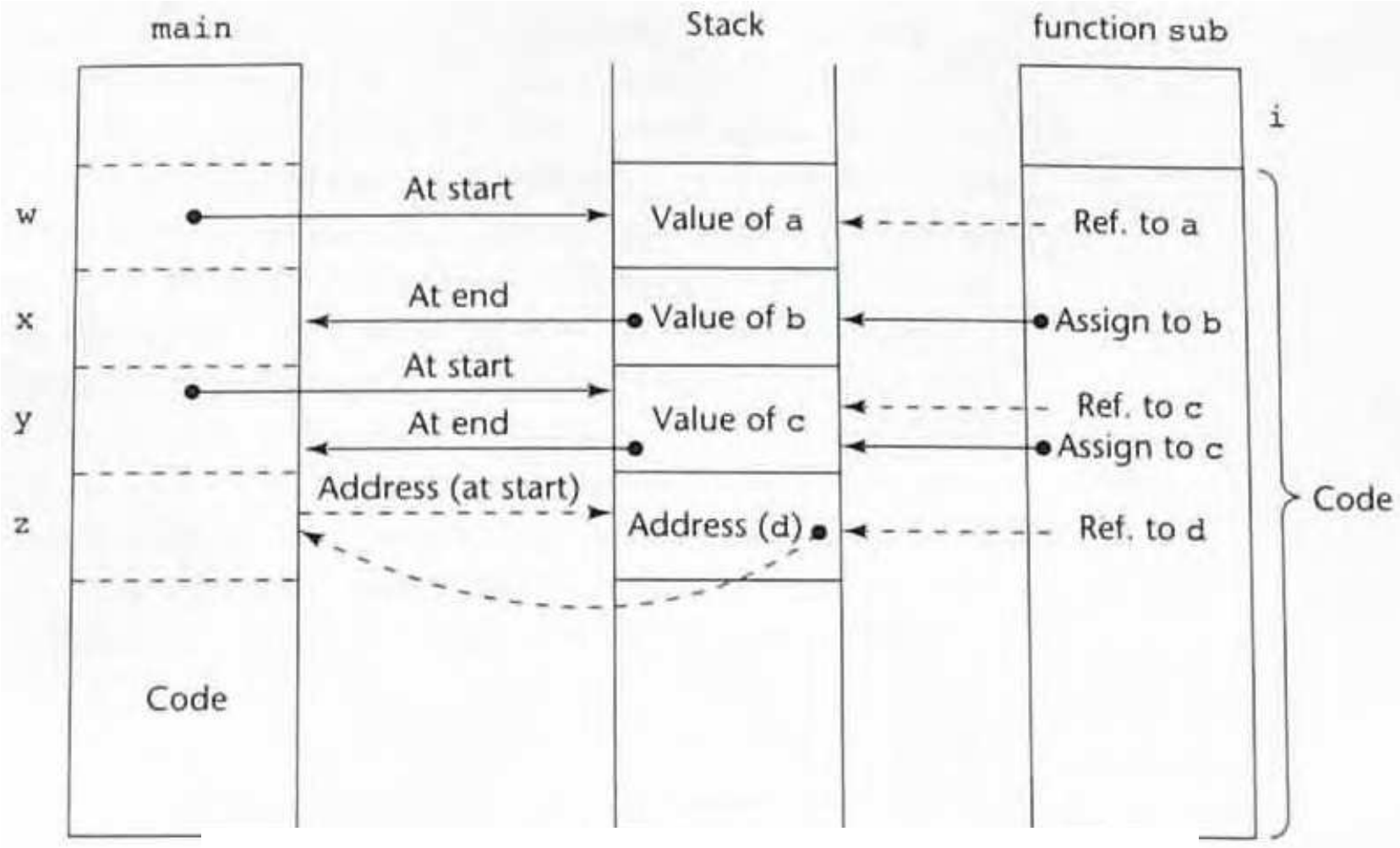
- Cách cài đặt thứ 2 của Inout Mode
- Truyền bằng 1 đường dẫn, có thể địa chỉ ô nhớ
- Tham số hình thức là pointer
- Hiệu quả hơn (không cần không gian lưu trữ)
- Khuyết điểm
  - Truy xuất chậm hơn (so sánh với truyền tham trị)
  - Có thể gặp vấn đề biệt danh (alias) không mong đợi bởi vì các truy xuất là không cục bộ. VD trong C

void fun(int &first, int &second) lúc gọi fun(total,total)

Cài đặt các cách truyền tham số

- Hầu hết các NNLT điều dùng stack để xây dựng cơ chế truyền tham số
- Truyền tham trị sẽ copy giá trị của tham số thực vào trong stack tương ứng giá trị của tham số hình thức
- Truyền tham trị-kết quả thì giá trị của tham số hình thức được lưu trong stack và sẽ trả về cho tham số thực
- Truyền quy chiếu là đơn giản nhất, chỉ cần lưu địa chỉ ô nhớ vào trong stack

Cài đặt các cách truyền tham số



Hàm Main gọi sub(w, x, y, z) : w truyền tham trị, x truyền kết quả, y truyền tham trị-kết quả, z truyền quy chiếu

Cách truyền tham số trong các NNLT

- Fortran
  - Luôn dùng mô hình Inout
  - Trước Fortran 77: truyền quy chiếu
  - Từ fortran 77 trở về sau: truyền kết quả
- C
  - Truyền tham trị
  - Truyền quy chiếu với tham số hình thức khai báo kiểu con trỏ
- C++
  - Truyền tham trị
  - Truyền quy chiếu với tham số hình thức khai báo kiểu con trỏ
  - Tham số đối tượng truyền quy chiếu
- Java
  - Tất cả tham số đều truyền tham trị
  - Tham số đối tượng truyền quy chiếu

Cách truyền tham số trong các NNLT

- Ada
  - Dùng 3 từ khóa để xác định cách truyền tham số : `in`, `out`, `in out`; mặc định là `in`
  - Có thể gán trị cho tham số hình thức được khai báo với `out` nhưng trị đó không được tham khảo, còn những tham số được khai báo với `in` thì không trả về trị; tham số với `in out` thì truyền tham trị và trả về kết quả
- C#
  - Mặc định là truyền tham trị
  - Truyền tham số được xác định trong cả tham số hình thức và tham số thực bởi từ khóa `ref`
- PHP: Giống như C#
- Perl: tất các các tham số thực điều được đặt sau @_

Kiểm tra kiểu các tham số

- Kiểm tra kiểu của các tham số là rất cần thiết (for reliability)
- FORTRAN 77 và original C: không kiểm tra
- Pascal, FORTRAN 90, Java, và Ada: luôn luôn kiểm tra kiểu
- ANSI C và C++: Tùy thuộc vào người dùng
  - Prototypes : khai báo hàm

  Double sin(x) Double sin (double x){….}

  double x; {…}
- Perl, JavaScript, và PHP thì không kiểm tra kiểu

Tham số là mãng nhiều chiều

- Nếu tham số của CTC là mãng nhiều và CTC và CT gọi CTC được dịch độc lập thì chương trình dịch cần khai báo kích thước của mãng để xây dựng các chỉ số index

Tham số mãng nhiều chiều: C và C++

- Yêu câu người dùng phải chỉ rõ số cột trong tham số hình thức đối với mãng 2 chiều
  - `void fun(int matrix[][10]);`
- CTC không được linh hoạt
- Giải pháp: dùng biến con trỏ trỏ đến mãng và kích thước của các chiều thì truyền bằng cách tham số khác ~ người dùng phải chỉ ra kích thước lưu trữ của mãng thông qua các tham số
  - VD `void(float *mat_ptr, int num_rows, int num_cols);`

Tham số là mãng nhiều chiều : Java và C#

- Mãng là 1 đối tượng, do đó tất cả các mãng đều 1 chiều nhưng từng phần tử có thể là mãng
- Mỗi mãng thừa kế 1 hằng số (`length` trong Java, `Length` trong C#) được xem là chiều dài của mãng ngay lúc khởi tạo

Chọn cách truyền tham số

- Hai cân nhắc quan trọng
  - Tính hiệu quả
  - Truyền một chiều hay truyền 2 chiều
- Nghịch lý
  - Người ta khuyên là nên hạn chế truy xuất các biến, tức là nên dùng truyền 1 chiều nhiều nhất có thể
  - Nhưng truyền tham quy chiếu là cách hiệu quả nhất

Tham số là tên của CTC

- Mội vài NNLT cho phép dùng tên của CTC như là một tham số
- VD hàm integral

```
procedure integrate(function (fun(x :
real) : real; lbound, rbound : real);
```

- C và C++: không hỗ trợ cơ chế dùng tên hàm như tham số

Tham số là tên của CTC - javaScript

```
function sub1(){
  var x;
  function sub2(){
    alert(x)
  };
  function sub3(){
    var x; x = 3;
    sub4(sub2);
  }
  function sub4(subx){
    var x; x= 4;
    subx();
  };
  x=1;
  sub3();
};
```

Giá trị của x là bao nhiêu 4 hay 1 trong ngôn ngữ phạm vi động và liên kết cạn và liên kết sâu

Chương trình con đa năng

- Hầu hết các NNLT điều có các phép toán đa năng
- Chương trình con đa năng là CTC có cùng tên với hàm có sẵn trong cùng một phạm vi
  - Tất cả các phiên bản đều có chung 1 protocol
- Trình biên dịch chọn phiên bản thích hợp dựa trên các tham số của hàm
- Ada, Java, C++, và C# cho phép người dùng viết nhiều phiên bản của CTC chùng tên nhau
- C++, Java, C#, và Ada cho phép thêm vào các CTC đa năng (VD toán tử)

Chương trình con đa năng (tt)

- VD ba hàm trả về trị tuyệt đối của một tham số

```
int MyAbs(int X) {
  return abs(X);
}
long MyAbs(long X){
  return labs(X);
}
double MyAbs(double X){
  return fabs(X);
}
```

  - `int a; long b; MyAbs(a); MyAbs(b)` : trình biên dịch dựa vào kiểu của tham số để xác định phiên bản thích hợp

Chương trình con chung

- CTC chung (*generic*) hay đa hình (*polymorphic*) là một tên CTC có thể chấp nhận các tham số có nhiều kiểu khác nhau
- CTC đa năng là trường hợp đặc biệt của CTC chung
- Các tham số chung dùng để mô tả các kiểu khác nhau gọi là tham số đa hình (*parametric polymorphism*)

VD tính đa hình của CTC: C++

- Định nghĩa 1 template

```
template <class Type>
Type max(Type first, Type second) {
   return first > second ? first : second;
}
```

- Template trên có thể đại diện cho phép toán so sánh lớn hơn “>” với tất các kiểu của khác nhau
  - `int a,b,c; char c,d,f;`
  - `C = max(a,b); f=max(d,e);`
- VD so sánh cho kiểu integer thông thường

```
int max (int first, int second) {
  return first > second? first : second;
}
```

VD tính đa hình của CTC: C++ (tt)

```
template <class type>
void generic_sort (Type list[], int len){
  int top, bottom;
  Type temp;
  for(top=0;top<len-2;top++)
    for(bottom=top+1;bottom<len-1;bottom++)
      if(list[top]>list[bottom]){
        temp = list[top];
        list[top] = list[bottom];
        list[top] = temp:
      }/*end of if*/
}/*end of generic_sort*/
```

Khi thiết kế hàm : các yếu tố

- Có cho phép hiệu ứng lề không?
  - Các tham số nên ở in-mode để giảm hiệu ứng lề (như Ada)
- Cho phép giá trị trả về có kiểu gì?
  - Hầu hết các NNLT điều giới hạn kiểu trả về
  - C cho phép trả về với bất cứ kiểu gì trừ kiểu mãng
  - C++ cũng giống như C nhưng bao gồm luôn cả kiểu do người dùng định nghĩa
  - Ada cho phép tất cả các kiểu
  - Java và C# không có hàm nhưng các **methods** có thể trả về bất kỳ kiểu gì?

Phép toán đa nghĩa: do người dùng cài đặt

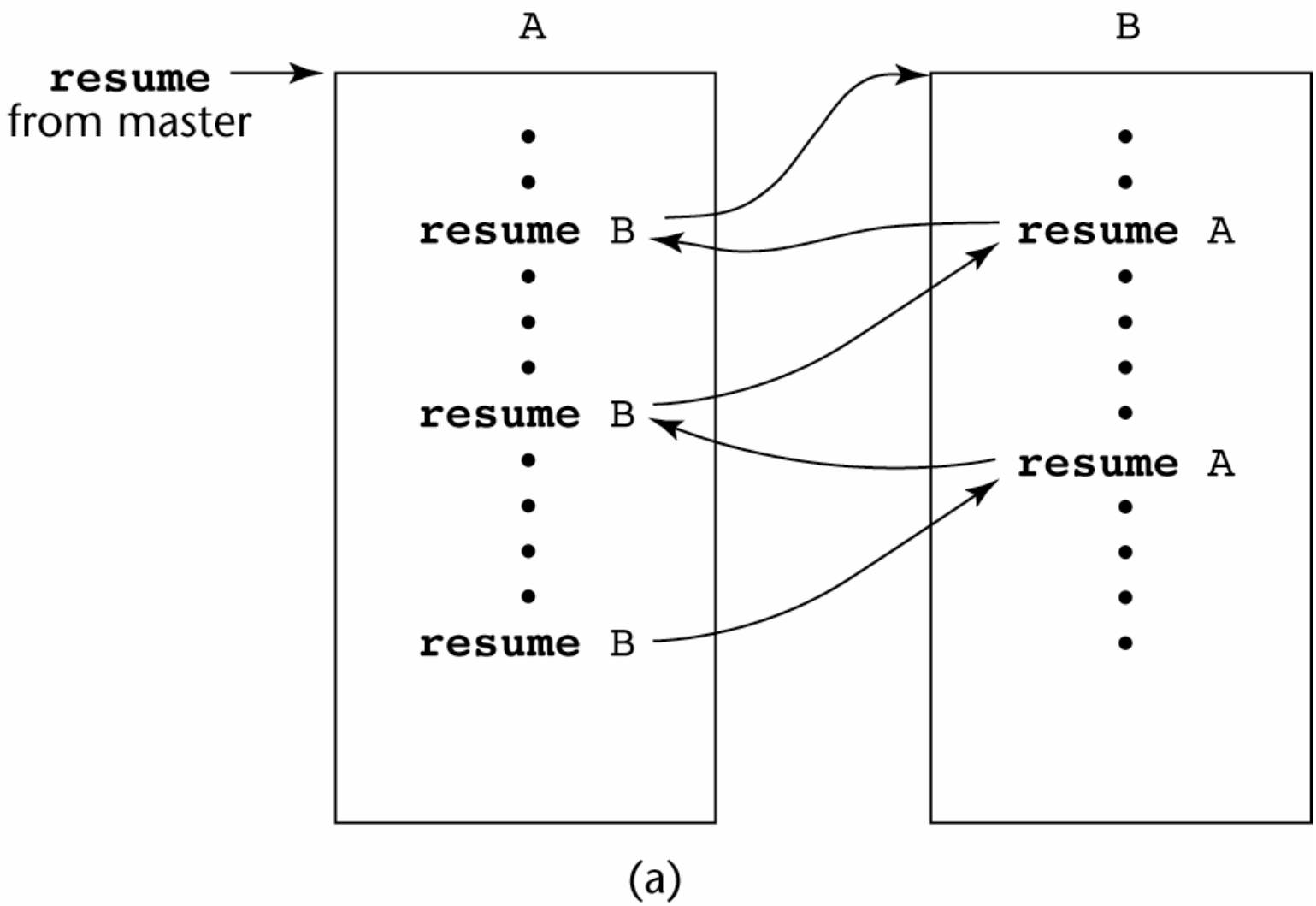
- Ada và C++ cho phép người dùng cài đặt các phép toán đa nghĩa
- VD trong Ada

```
Function “*”(A,B: in Vec_Type): 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 = a * b; -- a, b, and c are of type Vec_Type
```

Sự đang xen (Coroutines)

- Xự đang xen là một CTC có nhiều điểm vào (multiple entries) và điều khiển lẫn nhau
- Chương trình gọi (caller) và bị gọi (called) gọi đang xem lẫn nhau
- Còn được gọi là điều khiển đối xứng (symmetric control)
- Sự gọi đang xen được đặt tên là *resume*
- Sự đang xen có thể lập đi lập lại và có thể không dừng

Minh họa sự đang xen: trường hợp 1



Minh họa sự đang xen: trường hợp 2