

C语言程序设计

计算机科学与技术学院

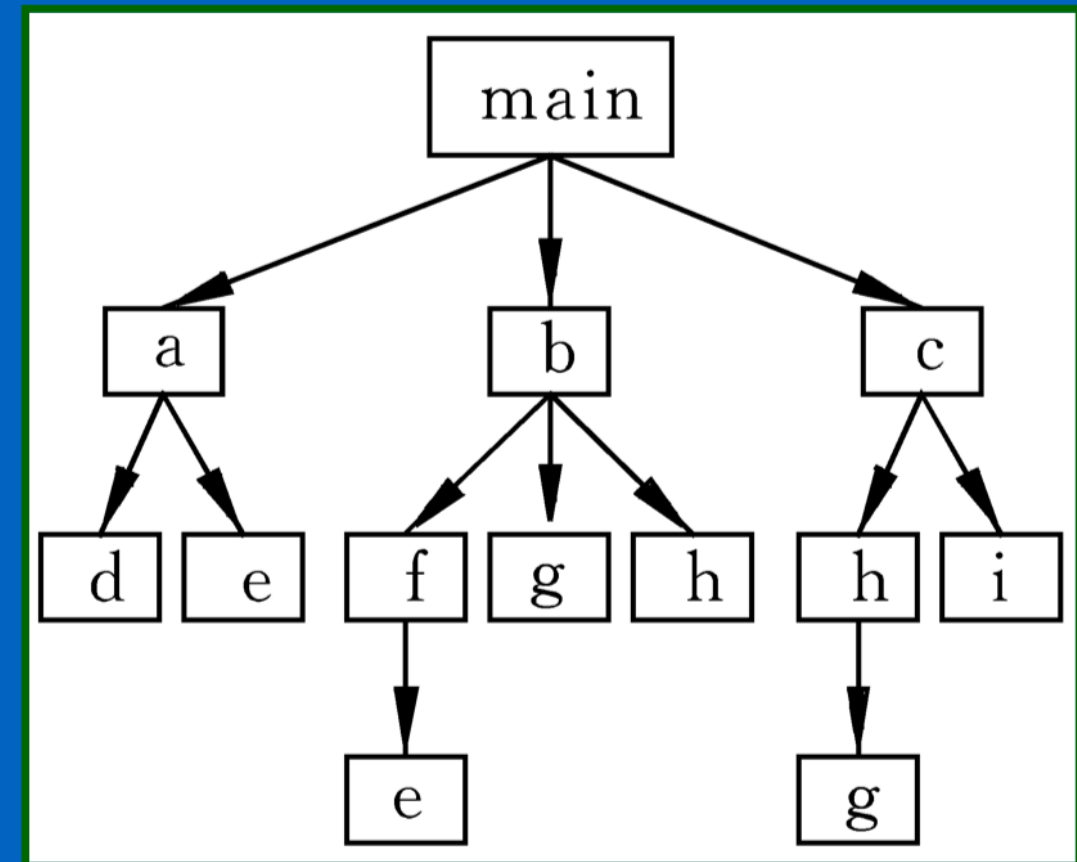
函数

- 库函数
- 自定义函数
 - 定义
 - 声明
 - 调用



模块化结构设计

- 一个较大的程序可分为若干个程序模块(module)
 - 每个模块实现一个特定的功能
 - 每个模块可由不同的程序员编写
- 在C语言中，模块通过函数(function)实现
 - 一个C程序必须有且只有一个名为main()的主函数
 - C程序的执行总是从main函数开始，在main中结束
 - C程序还可包括若干个其他函数
 - 一个函数可以被一个或多个函数调用任意多次
- 一个C程序可由若干源程序文件(source code)组成
 - 每个源程序文件由一系列数据类型定义、变量定义、函数定义等代码组成
 - 每个源程序文件可由不同程序员独立编写和编译



库函数与自定义函数

- C系统中函数分为两大类
 - 库函数（标准函数）
 - 自定义函数
- 库函数
 - 库函数是C系统为用户编写好的程序
 - C系统提供了丰富的库函数
- 自定义函数
 - 为实现特定功能，用户自行编写的函数
- 常用的库函数类别
 - `stdio.h`，输入输出库函数(`printf/scanf/getchar/putchar/gets/puts/fprintf/fscanf`)
 - `string.h`，字符串处理库函数(`strlen/strcmp/strcat/strcpy`)
 - `stdlib.h`，标准库函数(`exit/rand`)
 - `math.h`，数学库函数(`sqrt/power`)
 - `time.h`，时间库函数(`time`)
 - `malloc.h`，动态存储分配库函数
 -
- 每类库函数都定义了自己专用的常量、符号、数据类型、函数接口和宏等
- 使用相应库函数，要包含其头文件的预处理命令

常用函数

- math.h

- 平方根函数sqrt(x)
- 绝对值函数fabs(x)
- 幂函数pow(x,n)
- 以e为底的指数函数exp(x)
- 以e为底的对数函数log(x)

- stdlib.h

- 随机数
- int rand(void);
- #define RAND_MAX 0x7fff // 0x7fff == 32767
- void srand(unsigned int _Seed);

- time.h

- time(NULL)

自定义函数

- 为实现特定功能，用户自行编写的函数
- 求两个数的和

```
#include <stdio.h>
int main (){
    int a, b;

    scanf("%d,%d",&a,&b);
    printf("%d", a + b);

    return 0;
}
```

```
#include <stdio.h>
int add (int, int);
int main(){
    int a,b;

    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b));

    return 0;
}

//自定义函数
int add(int x, int y){
    return x + y;
}
```

函数的定义、声明和调用

- **定义**：确立函数功能

```
<返回值类型>  函数名 (<形参列表>) {  
    函数体  
}
```

- **声明**：函数原型说明

```
<返回值类型>  函数名 (<形参列表>) ;
```

- **调用**：执行函数所定义的功能

```
函数名(实参类型列表);
```

```
#include <stdio.h>  
  
int add (int, int);    //函数声明  
  
int main(){  
    int a,b;  
  
    scanf("%d,%d",&a, &b);  
    printf("%d", add(a, b));    //函数调用  
  
    return 0;  
}  
  
//函数定义-起始  
int add(int x, int y){  
    return x + y;  
}  
//函数定义-结束
```

函数的声明

- 声明：函数原型说明

- 把函数的名字、函数类型以及形参的类型、个数和顺序通知编译系统
- 检查手段：调用该函数时，系统按此进行对照检查
- 标准库函数的函数原型都在头文件中提供
- 用户自定义函数，程序员应该在源代码中说明

- 一般形式

<返回值类型> 函数名 (<形参列表>) ;

```
#include <stdio.h>
int add (int, int);    //函数声明
int main(){
    int a,b;

    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b));    //函数调用

    return 0;
}

//函数定义-起始
//int Add(float x, int y){  出错：与声明不一致
int add(int x, int y){
    return x + y;
}
//函数定义-结束
```


函数的声明：是否必要？

- 函数定义在后、调用在前：必须声明

```
#include <stdio.h>
int add(int, int);    //函数声明，不可省略

int main(){
    int a,b;
    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b));    //函数调用在前
    return 0;
}

int add(int x, int y){    //函数定义在后
    return x + y;
}
```

- 函数定义在前、调用在后：不必声明

```
#include <stdio.h>
int add(int, int);    //函数声明，可省略

int add(int x, int y){    //函数定义在前
    return x + y;
}

int main(){
    int a,b;
    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b));    //函数调用在后
    return 0;
}
```

- 通常也声明

函数的声明：知名度？

- 面向所有函数

```
#include <stdio.h>
int add(int, int);    //面向所有函数

int main(){
    int a,b;
    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b)); //main可调用add
    return 0;
}

int add(int x, int y){
    return x + y;
}

int sum(int x, int y)
    return add(x + y); //sum可调用add
}
```

- 面向所在函数

```
#include <stdio.h>

int main(){
    int add(int, int);    //面向所在函数
    int a,b;
    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b)); //main可调用add
    return 0;
}

int add(int x, int y){
    return x + y;
}

int sum(int x, int y)
    return add(x + y); //出错：sum不可调用add
}
```

- 通常也声明

函数的定义

- 定义：确立函数功能
 - 函数名、返回值类型、形参及其类型
 - 函数体：完成具体功能的代码块
- 一般形式

```
<返回值类型>  函数名 (<形参列表>) {  
    函数体  
}
```

```
#include <stdio.h>  
int add(int, int);  
int main(){  
    int a,b;  
    scanf("%d,%d",&a, &b);  
    printf("%d", add(a, b));  
    return 0;  
}
```

```
//函数定义起始  
int add(int x, int y){  
    return(x + y);  
}  
//函数定义结束
```

函数的定义: 不能嵌套定义

- 在一个函数的函数体内部不得再定义其他函数

```
int f1(int a,int b)
{
    ....
    int f2(int i,int j) //错误
    {
        return i+j;
    }
    ....
}
```

```
int f1(int a,int b)
{
    int k;
    ....
    k = f2(a,b);
    ....
}

int f2(int i,int j)
{
    return i+j;
}
```

函数的定义: 返回值和返回值类型

- 函数的功能通常是为了完成某些特定操作, 可得到一个结果 (即返回值)
- 函数的返回值: 在函数调用结束后, 向调用者 (Caller) 传出给调用者的人
 - 通过带有表达式的返回语句来结束函数调用
 - `return 表达式; 或 return (表达式);`
- 返回值类型: 返回值的数据类型

```
#include <stdio.h>
int add(int, int);
int main(){
    int a,b;
    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b));
    return 0;
}
//返回值类型: int
int add(int x, int y){
    //返回值: return之后的表达式的值
    return x + y;
}
```

函数的定义: 能否具有多个返回值?

- C程序中, 一个函数最多有一个返回值
 - 在一个函数中, 可以有一个或多个return语句
 - 执行一条return语句后即结束函数调用 (其他语句不再继续执行)

```
#include <stdio.h>
int add(int, int);
int fun(int, int, int);

int main(){
    int a,b,c;
    scanf("%d,%d,%d",&a, &b, &c);
    printf("%d", add(a, b));
    printf("%d", fun(a, b, c));

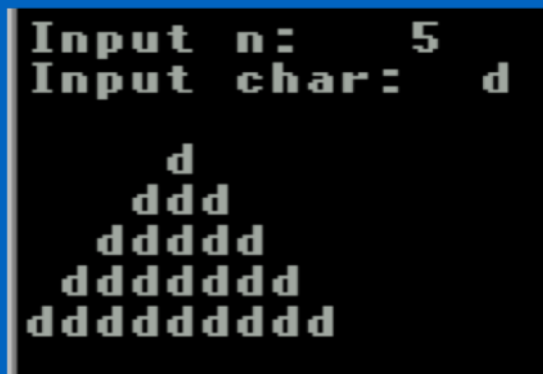
    return 0;
}
```

```
int add(int x, int y){
    //结束函数调用, 将返回值传递给main
    return x + y;
    //以下任何语句都没有执行机会
    return x - y;
}

int fun(int x, int y, int c){
    //以下两个return, 最终只能执行一个
    if (c > 0)
        return x + y;
    else
        return x - y;
}
```

函数的定义：无返回值情况

- 函数无返回值时
 - 返回值类型为空类型：void
 - 对应的返回值语句：return;
 - 如果一个函数中无return语句，遇到函数体的右花括号}自动return（即可省略右花括号前的return;）



```
Input n: 5
Input char: d

  d
 ddd
 dddd
 ddddd
 ddddddd
 dddddddd
```

```
#include <stdio.h>
void printStart(int, char);
int main(){
    int n;
    char ch;
    printf("Input n: ");
    scanf("%d", &n);
    printf("Input char: ");
    scanf(" %c", &ch);
    printf("\n");
    printStart(n, ch);
    return 0;
}

//定义无返回值的函数
void printStart(int n, char c){
    int i, j;
    for(i = 1; i <= n; i++){
        for(j = n - 1; j >= i; j--){
            printf(" ");
        }
        for(j = 1; j <= 2 * i - 1; j++){
            printf("%c", c);
        }
        printf("\n");
    }
    return; //此语句亦可省略
}
```

函数的定义: 返回值类型与return语句中的结果不一致?

- 最终类型是函数定义中的返回值类型
1. 首先计算return语句中<表达式>的值
 2. 然后将<表达式>的类型转换为函数定义中的返回值类型

```
#include <stdio.h>
int add(float, float);

int main(){
    int a,b;
    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b));

    return 0;
}
```

```
//函数返回值类型为int
int add(float x, float y){

    // 表达式的结果为float类型
    return x + y;

    //return (int)(x + y); 可读性更好
}
```


函数的定义：形式参数(formal parameter)

- 形参：定义函数时,函数名之后的变量类型和名称列表
- 形参是函数被调用时传入并处理的各个数据的别名
- 函数若无形参，则应在,函数名后使用void，如 void fun(void);
- 形参的生命周期：
 - 函数未调用时：形参不占有内存单元
 - 函数被调用时：使用实际参数对形参进行初始化（按参数列表中相反顺序入栈）
 - 分配内存单元
 - 复制实际参数的值
 - 函数调用结束后：形参所占有的内存单元将被释放(出栈)

```
#include <stdio.h>

int add(int, int);

//形参y: add函数中，第一个接收的值的别名
//形参x: add函数中，第二个接收的值的别名
int add(int x, int y){
    return x + y;
}

void printStart(void){ //无参函数定义
    int i;
    for(i = 1; i <= 10; i++){
        printf("*");
        printf("\n");
    }
}

int main(){
    int a,b;
    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b));
    printStart();//无参函数调用

    return 0;
}
```

函数的调用

- 调用：执行函数所定义的功能
 - 通过引用函数名转到相应模块
 - 传递实参并执行
 - 模块结束时返回
 - 如有返回值，则将返回值传回调用者
- 函数名(实参类型列表);

```
#include <stdio.h>
int add (int, int);    //函数声明
int main(){
    int a,b;

    scanf("%d,%d",&a, &b);
    printf("%d", add(a, b));    //函数调用

    return 0;
}

//函数定义-起始
int add(int x, int y){
    return x + y;
}
//函数定义-结束
```

函数的调用：实际参数(actual parameter)

- 实参：调用函数时,函数名之后的变量列表
- 实参是调用函数 传递给 被调用函数 的、需要实际处理的数据
 - 具有确定的值
 - 可以是常量、变量或表达式
- 函数被调用时：使用实际参数对形参进行初始化（按参数列表中相反顺序入栈）
 - 为形参分配内存单元
 - 复制实际参数的值

```
#include <stdio.h>
int add(int, int);

//形参y: add函数中, 入栈参数1的副本的别名
//形参x: add函数中, 入栈参数2的副本的别名
int add(int x, int y){
    return x + y;
}

int main(){
    int a,b;
    scanf("%d,%d",&a, &b);
    //实参b : main函数中, 入栈参数1
    //实参a : main函数中, 入栈参数2
    printf("%d", add(a, b));
    return 0;
}
```

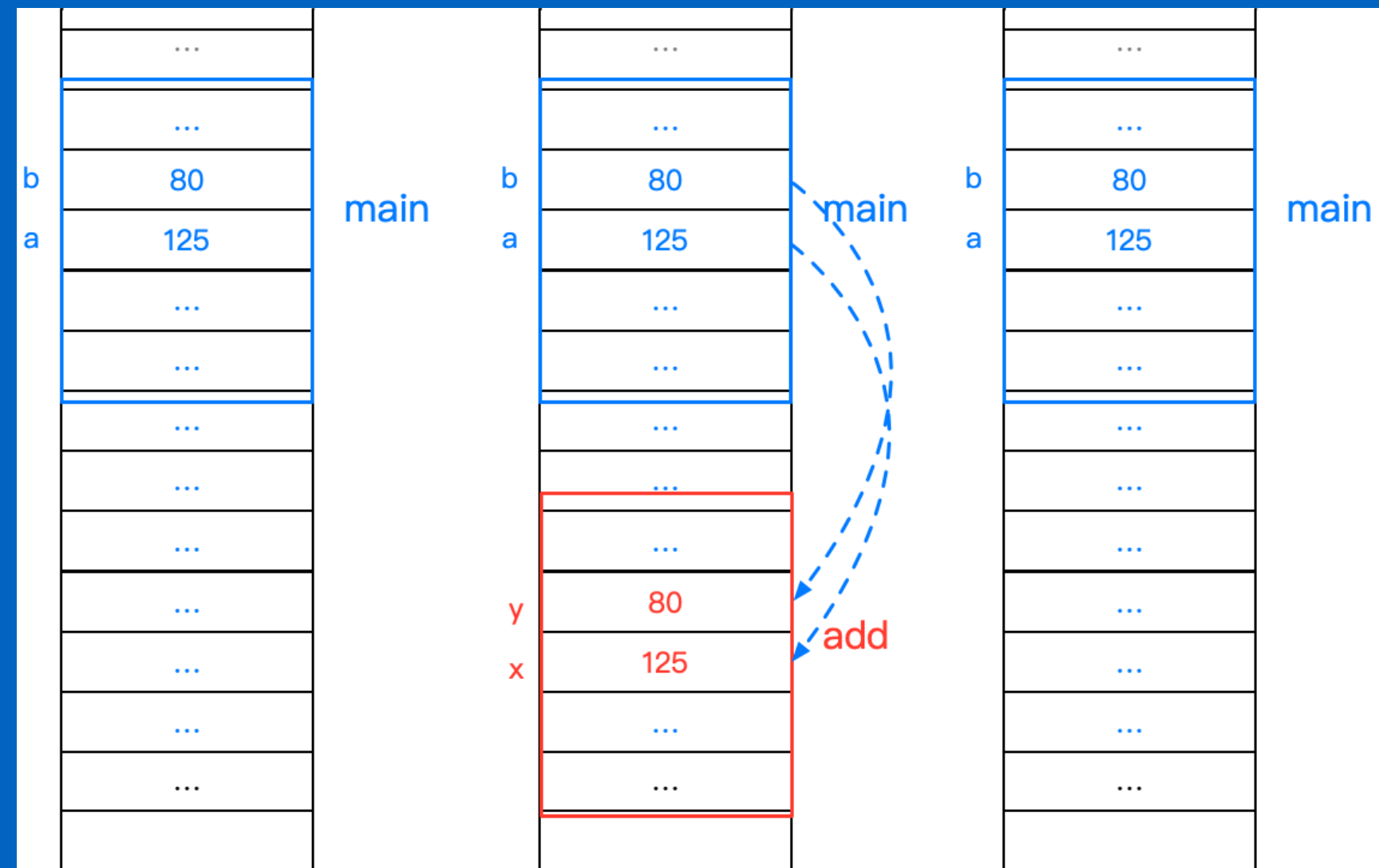
函数的调用：调用过程

```
#include <stdio.h>
int add(int, int);
```

```
//形参y: add函数中, 入栈参数1的副本的别名
//形参x: add函数中, 入栈参数2的副本的别名
```

```
int add(int x, int y){
    return x + y;
}
```

```
int main(){
    int a,b;
    scanf("%d,%d",&a, &b);
    //实参b : main函数中, 入栈参数1
    //实参a : main函数中, 入栈参数2
    printf("%d", add(a, b));
    return 0;
}
```



函数的调用：实际步骤举例(C与汇编)

```
int add ( int x, int y ) {  
    return x+y;  
}
```

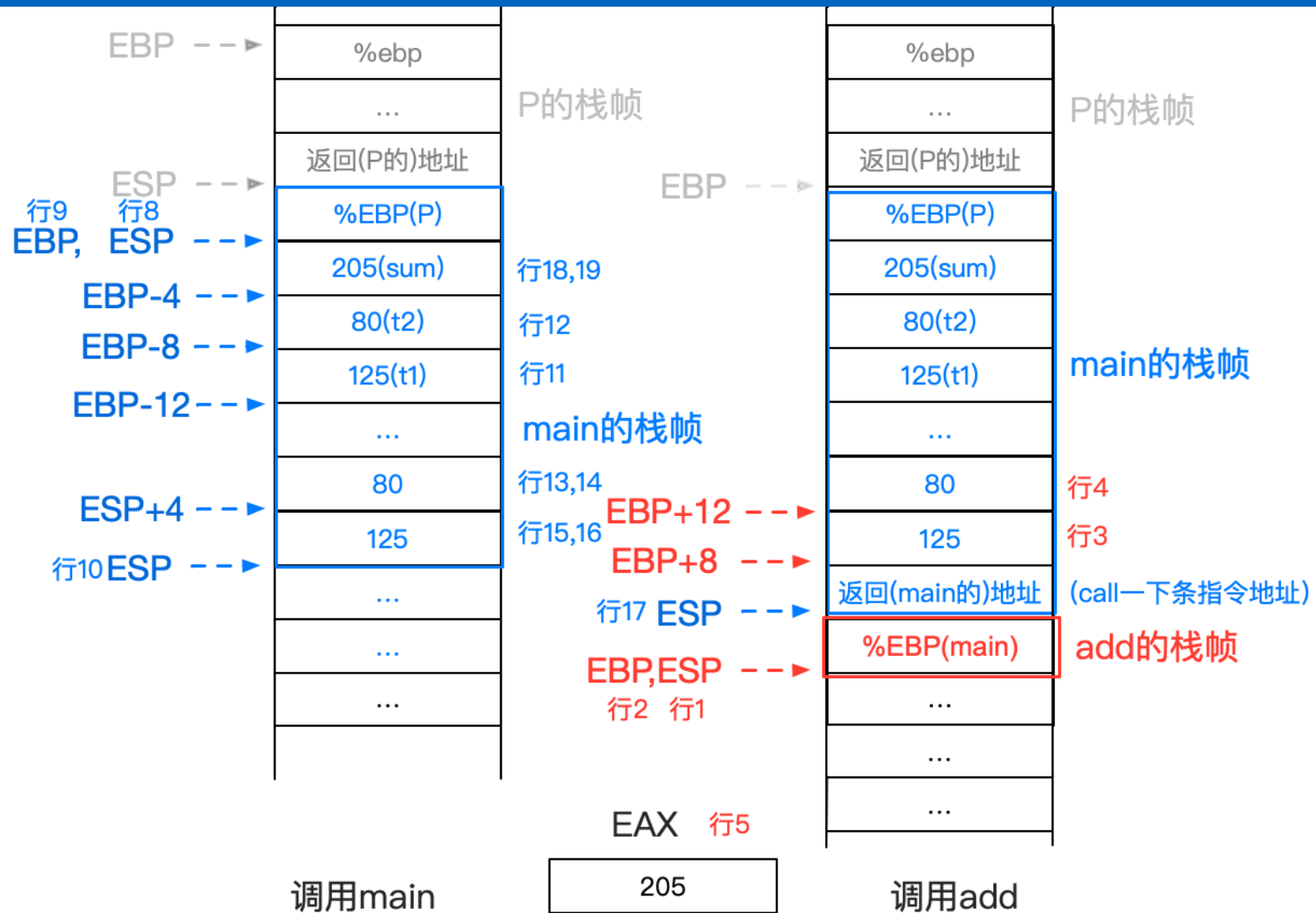
```

graph TD
    P -- dashed --> main
    main -- dashed --> add
    main -- dashed --> P

```

add:

```
main:
;准备阶段
8:pushl %ebp
9:movl %esp, %ebp
10:subl $24, %esp
;分配局部变量
11:movl $125, -12(%ebp)
12:movl $80, -8(%ebp)
;准备入口参数
13:movl -8(%ebp), %eax
14:movl %eax, 4(%esp)
15:movl -12(%ebp), %eax
16:movl %eax, (%esp)
17:call add
;准备返回参数(总在EAX中)
18:movl %eax, -4(%ebp)
19:movl -4(%ebp), %eax
;结束阶段
20:leave
21:ret
```



函数的调用：形参和实参的关系

- 实参和形参是属于不同函数的参数
- 在内存中实参和形参各自占有不同单元
- 形参与实参的个数相同，且类型一致
- 允许函数的实参与形参同名，此时它们亦是不同的变量
- 函数调用时，通过实参向形参传递数据（按参数列表中相反顺序入栈）
- 函数实参与函数形参的数据传递是单向的
- 函数调用结束后：形参所占有的内存单元将被释放（出栈）

```
#include <stdio.h>
int add(int, int);

//形参y: add函数中，入栈参数1的副本的别名
//形参x: add函数中，入栈参数2的副本的别名

int add(int a, int b){
    return a + b;
}

int main(){
    int a,b;
    scanf("%d,%d",&a, &b);
    //实参b : main函数中，入栈参数1
    //实参a : main函数中，入栈参数2
    printf("%d", add(a, b));
    return 0;
}
```

函数的调用：调用方式

- 函数语句：把函数调用作为一个语句
- 函数表达式：函数调用出现在一个表达式中、这种表达式称为函数表达式，此时要求函数必须是有返回值的
- 函数参数：函数调用作为一个函数的实参，把函数的返回值作为实参传递，此时要求函数必须是有返回值的

```
#include <stdio.h>

int add(int x, int y){
    return x + y;
}

void printXY(int x, int y){
    printf("%d,%d", x, y);
    return;
}

int main(){
    int a, b, c;
    scanf("%d,%d,%d",&a, &b, &c); //语句方式调用
    printXY(a, b); //语句方式调用
    printf("%d", add(a, b)); //函数参数
    printf("%d", add(a, b)+c); //函数表达式
    printf("%d", add(add(a, b), c)); //函数参数
    a = add(a, b); //函数表达式
    printf("%d", a = add(a, c)); //函数表达式

    return 0;
}
```

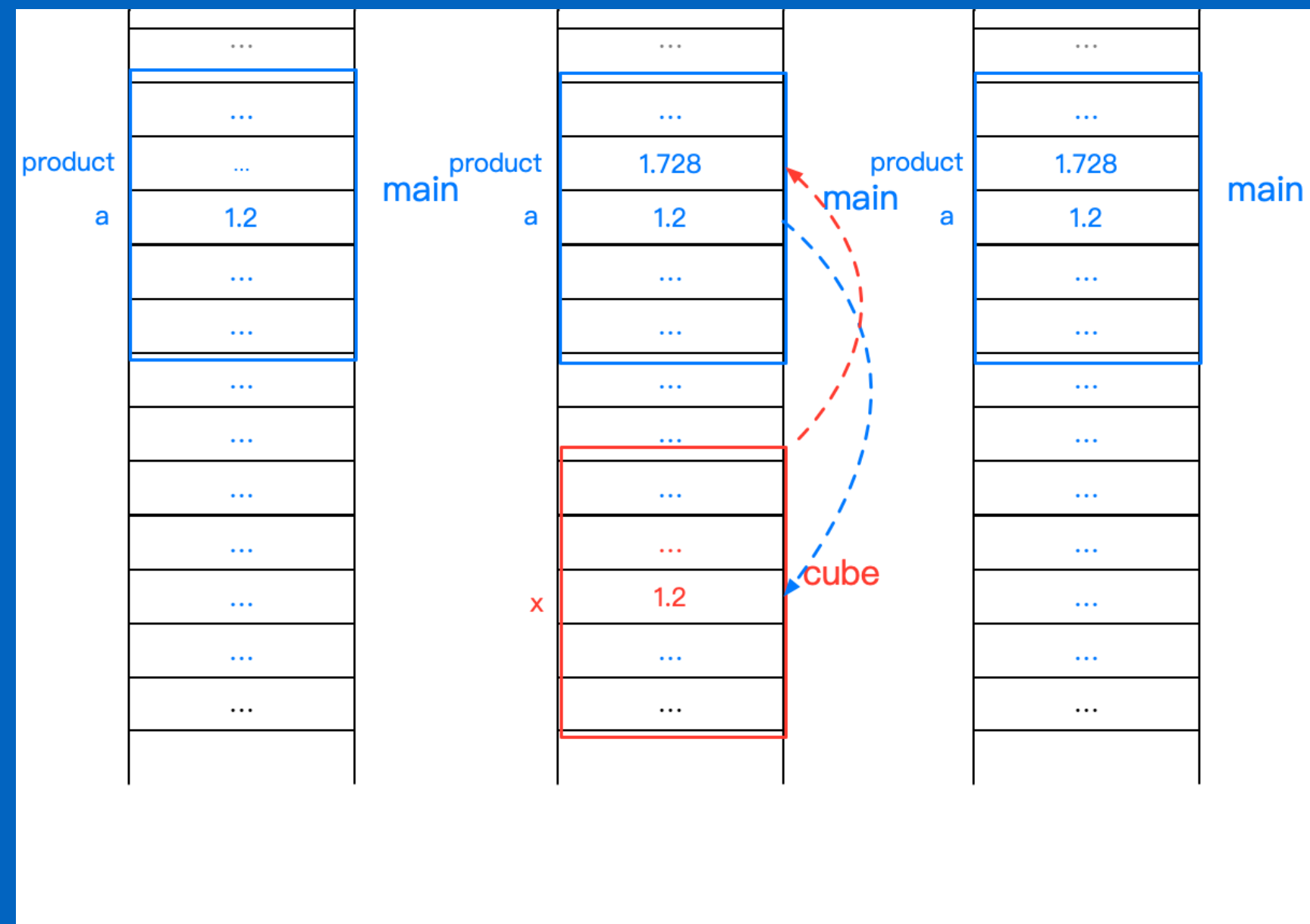
函数的调用：传值调用

- 实参复制一个副本给形参，使形参获得了对应的实参的值
- 例： 计算x的立方

```
#include <stdio.h>
```

```
float cube(float x){  
    return x * x * x;  
}
```

```
int main(){  
    float a, product;  
    printf("Please input value of a:");  
    scanf("%f", &a);  
    product = cube(a);  
    printf("Cube of %.4f is %.4f\n",a,product);  
  
    return 0;  
}
```



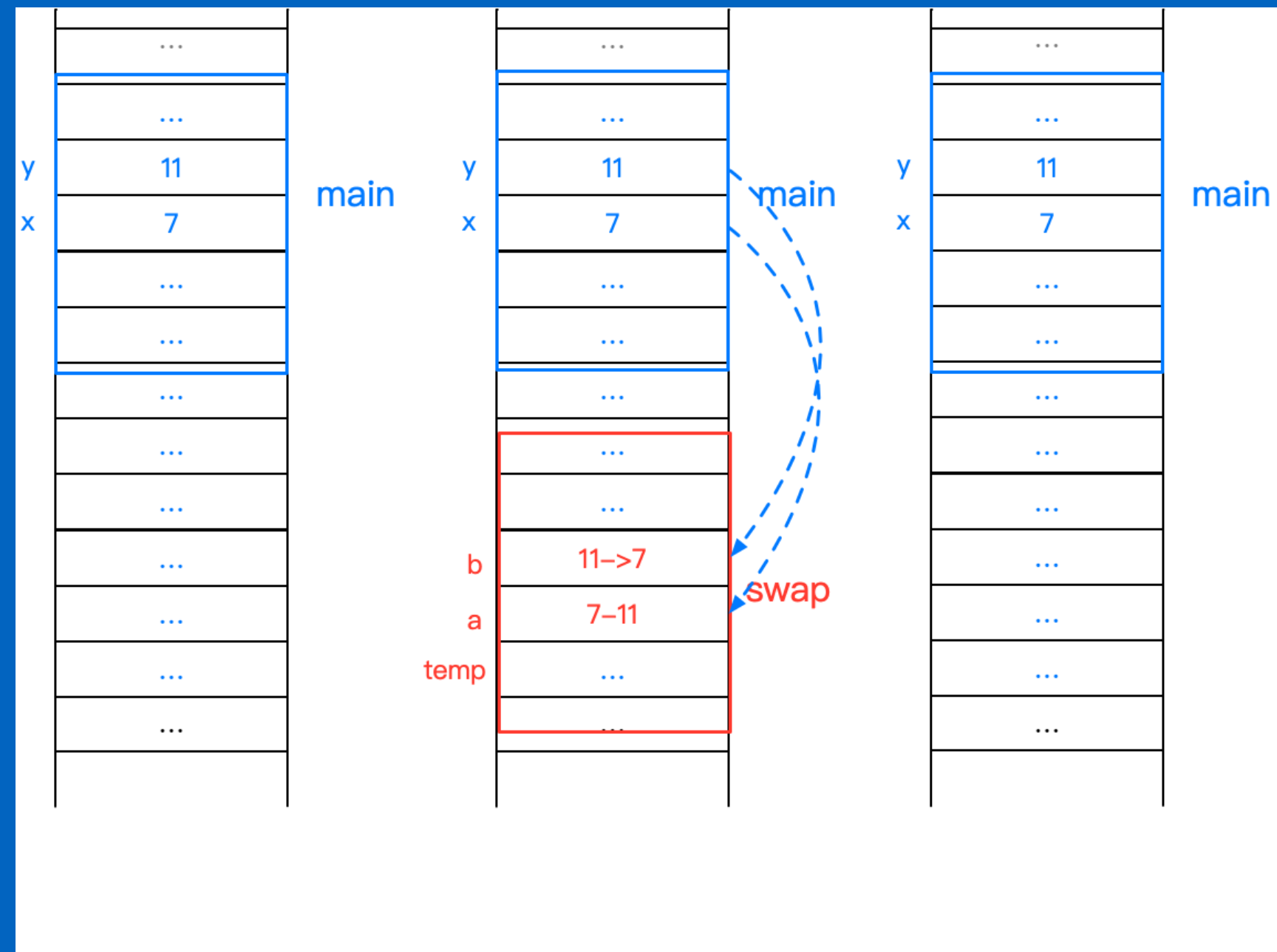
函数的调用：传值调用

- 实参复制一个副本给形参，使形参获得了对应的实参的值
- 例： 交换两个数

```
#include <stdio.h>
```

```
void swap(int a,int b){  
    int temp;  
    temp = a;  
    a = b;  
    b = temp;  
}
```

```
int main(){  
    int x = 7,y = 11;  
    printf("x=%d,\ty=%d\n",x,y);  
    printf("swapped:\n");  
    swap(x,y);  
    printf("x=%d,\ty=%d\n",x,y);  
  
    return 0;  
}
```



函数的调用：传值调用

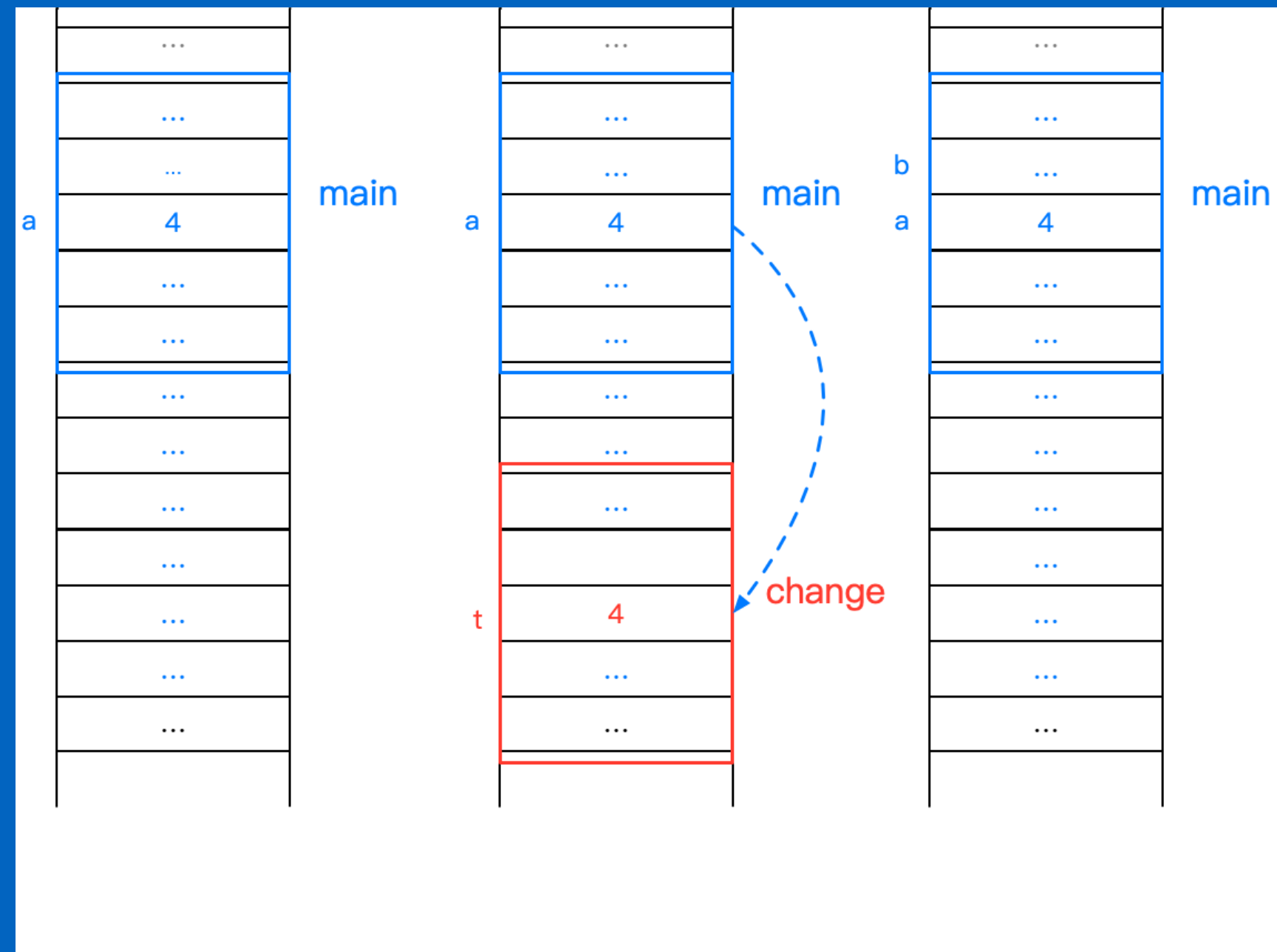
- 使用函数change把main中的变量a的值加 1

```
#include <stdio.h>
```

```
void change( int );
```

```
int main(){  
    int a;  
    a = 4;  
    change(a);  
    printf(“%d”,a);  
    return 0;  
}
```

```
void change( int t ){  
    a = t + 1; //change函数不认识a  
    t = t + 1; //t的变化无法反馈给a  
}
```



函数的调用：传变量的地址？

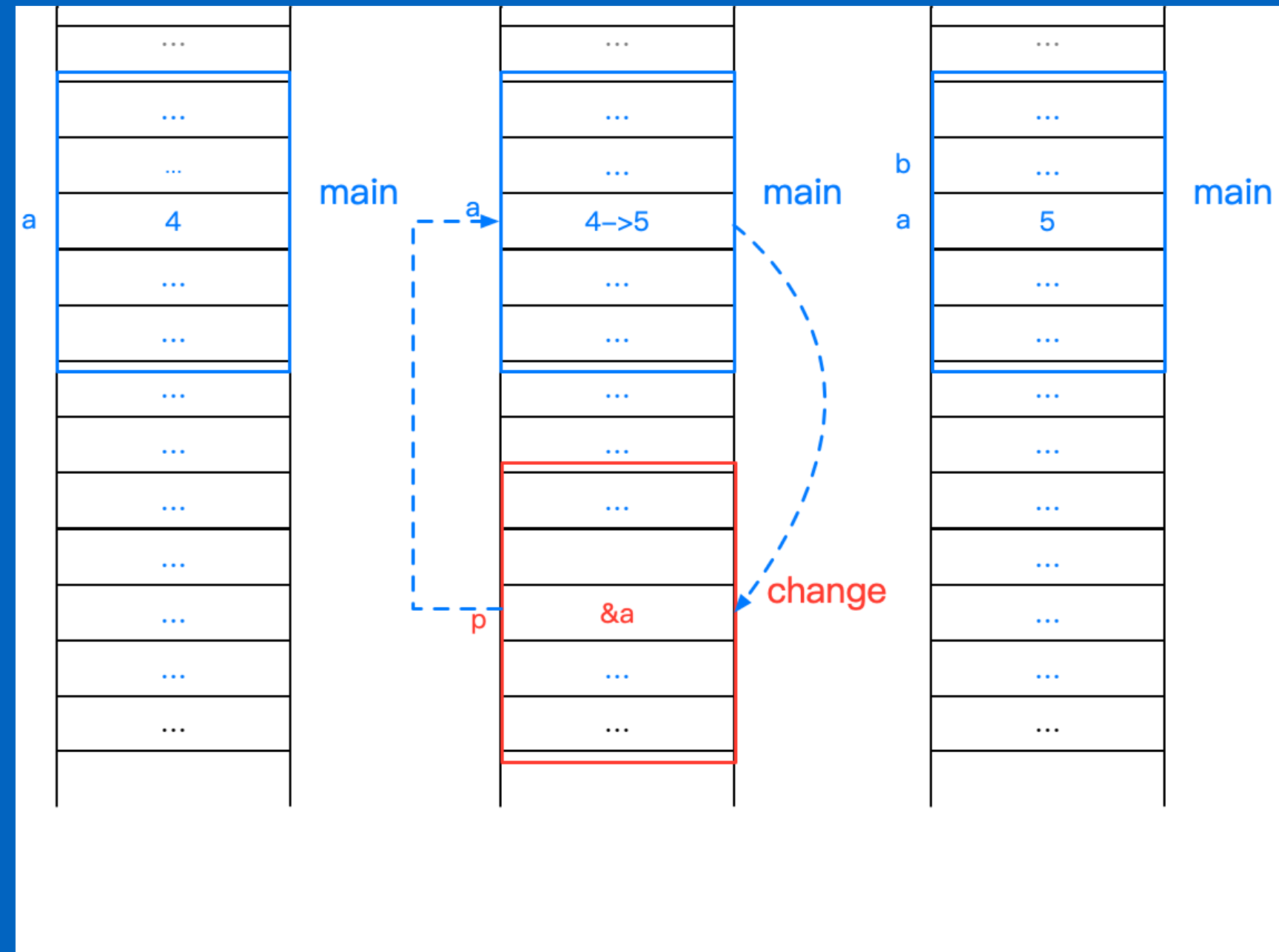
- 使用函数change把main中的变量a的值加 1

```
#include <stdio.h>
```

```
void change( 地址 );
```

```
int main(){  
    int a;  
    a = 4;  
    change(a的地址);  
    printf("%d",a);  
    return 0;  
}
```

```
void change( 地址 p ){  
    地址p 指向的变量值 ++  
}
```



函数的调用：传变量的地址？

- 使用函数change把main中的变量a的值加 1

```
#include <stdio.h>

void change( 地址 );

int main(){
    int a;
    a = 4;
    change(a的地址);
    printf("%d",a);
    return 0;
}

void change( 地址 p ){
    地址p指向的变量值 ++ ;
}
```

```
#include <stdio.h>

void change( int * );

int main(){
    int a;
    a = 4;
    change(&a);
    printf("%d",a);
    return 0;
}

void change( int * p ){
    (*p) ++;
}
```

函数的调用：传址调用

- 将实参的地址值传递给形参的指针，使形参指针指向实参变量。
- 要求实参是地址值，形参为相同类型的指针名
- 调用时，用实参地址值初始化形参的指针

特点：

- (1) 通过形参指针来改变调用函数所指向的变量（即实参）的值。
- (2) 由于传递的是变量的地址值，对一些复杂类型的变量具有较高运行效率。

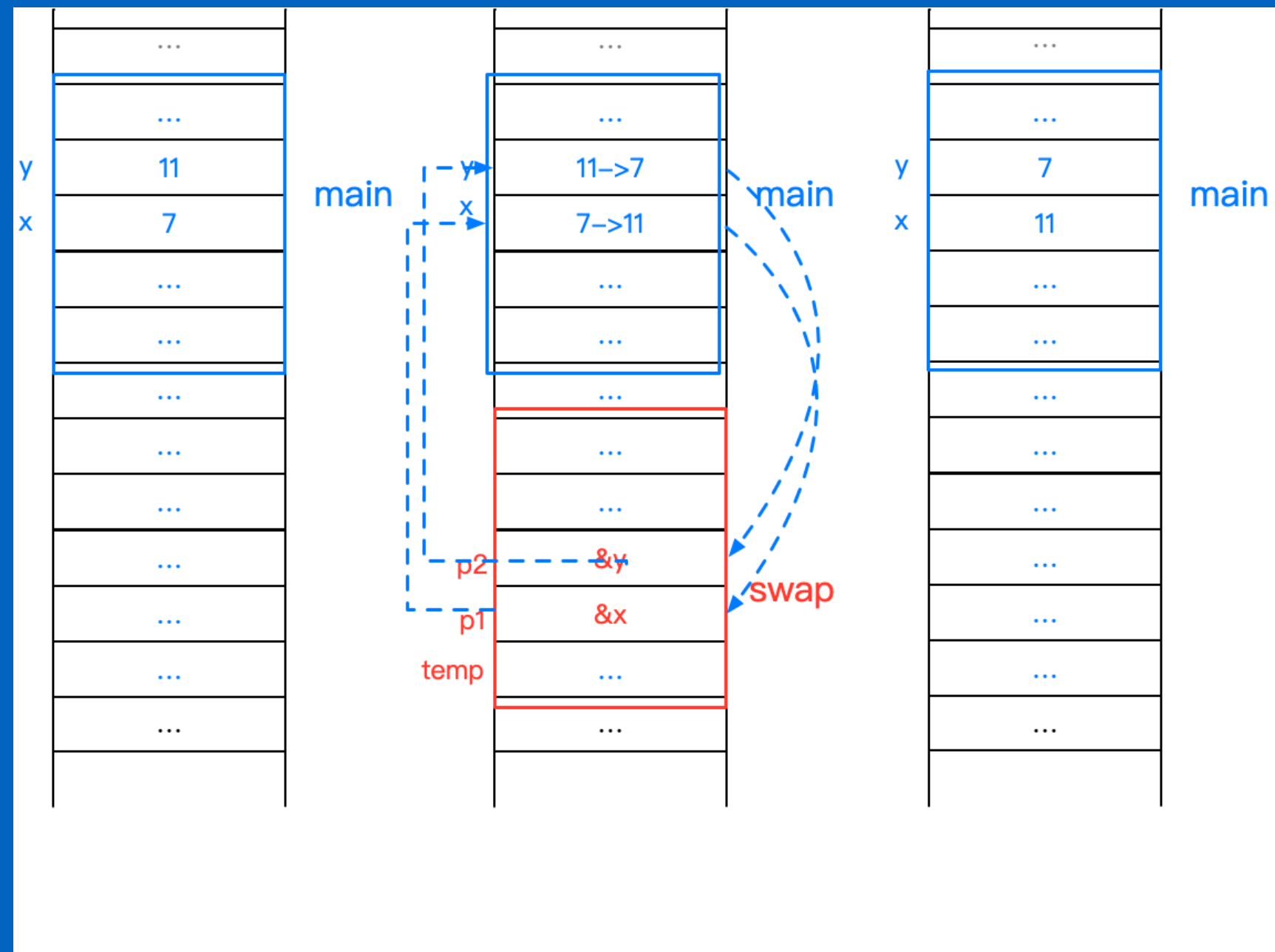
函数的调用：传址调用

- 将地址值传递给形参
- 例： 交换两个数

```
#include <stdio.h>
```

```
void swap(int *p1, int *p2){  
    int p;  
    //p=x;x=y;y=p;错误  
    p = *p1;  
    *p1 = *p2;  
    *p2 = p;  
}
```

```
int main(){  
    int x,y;  
    scanf("%d,%d",&x,&y);  
    printf("x=%d,y=%d\n",x,y);  
    printf("swapped:\n");  
    swap(&x, &y);  
    printf("x=%d,y=%d\n",x,y);  
  
    return 0;  
}
```



计算圆周率(应用蒙特卡洛算法思想)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

double countPI(int n){
    int i = 0, count = 0;
    double x = 0, y = 0;
    srand((unsigned)time(NULL));
    for (i = 0; i < n; i++){
        x = rand() / (double)(RAND_MAX);
        y = rand() / (double)(RAND_MAX);
        if ((x * x) + (y * y) < 1)
            count++;
    }
    return 4.0 * (count / n);
}

int main(){
    double pi = 0;
    clock_t start, finish;
    start = clock();

    pi = countPI(100000);
    printf("%lf\n", pi);

    finish = clock();
    printf( "%f seconds\n",
        (double)(finish - start) / CLOCKS_PER_SEC );

    return 0;
}
```

