4-4 Inter-Function Communication

Although the calling and called functions are two separate entities, they need to communicate to exchange data. The data flow between the calling and called functions can be divided into three strategies: a downward flow, an upward flow, and a bi-directional flow.

Topics discussed in this section:

Basic Concept
C Implementation

Note

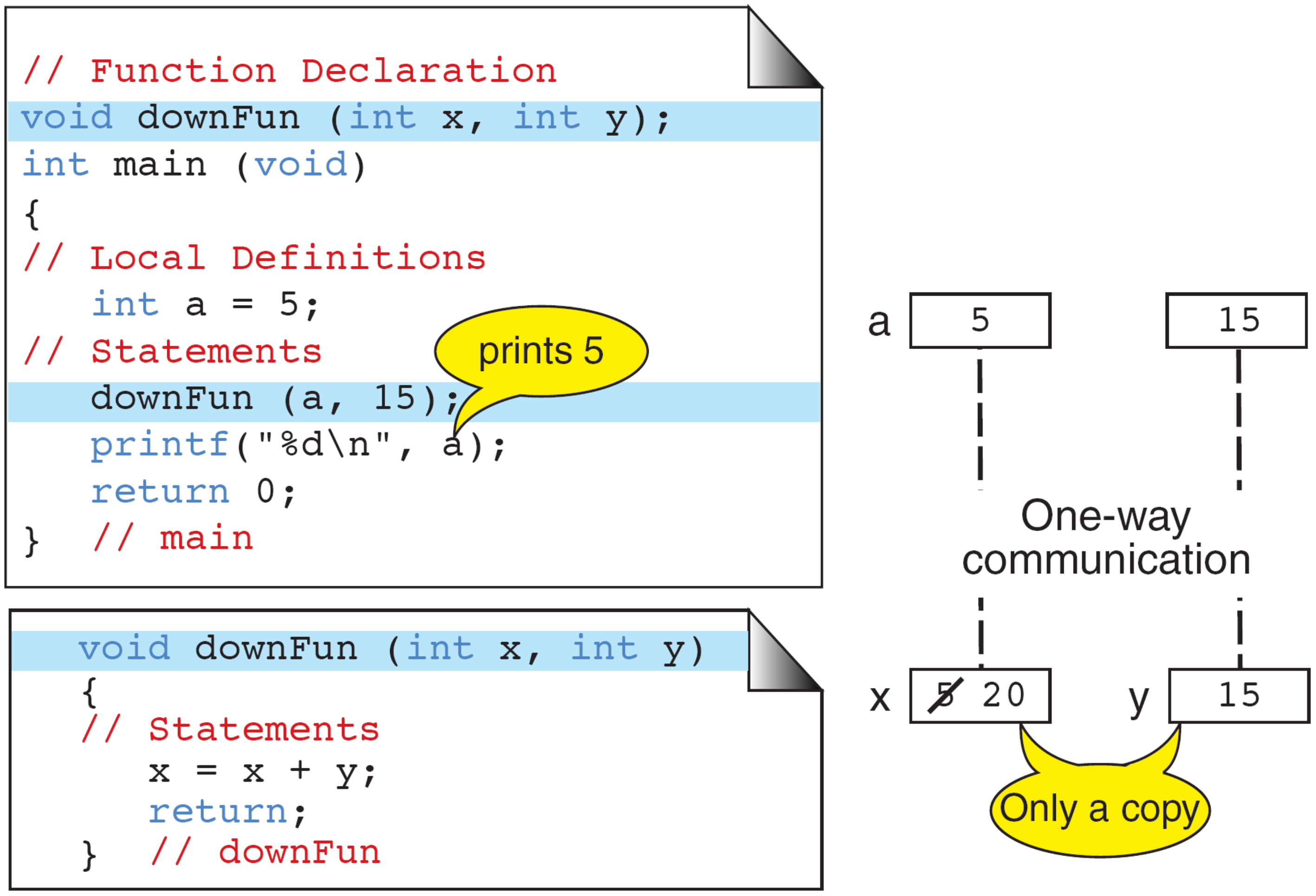
The C language uses only call by value and return to achieve different types of communications between a calling and a called function.

Call by value: value of parameter is passed, so changes made to parameter in function are not reflected in the calling function.

```c
int main (void)
{
   int a;
   ...
   downFun (a, 15);
   ...
}  // main
```

```c
void downFun (int x, int y)
{
   ...
   return;
}  // downFun
```

FIGURE 4-17 Downward Communication in C



```
// Function Declaration
void downFun (int x, int y);
int main (void)
{
// Local Definitions
   int a = 5;
// Statements
   downFun (a, 15);
   printf("%d\n", a);
   return 0;
}  // main
```

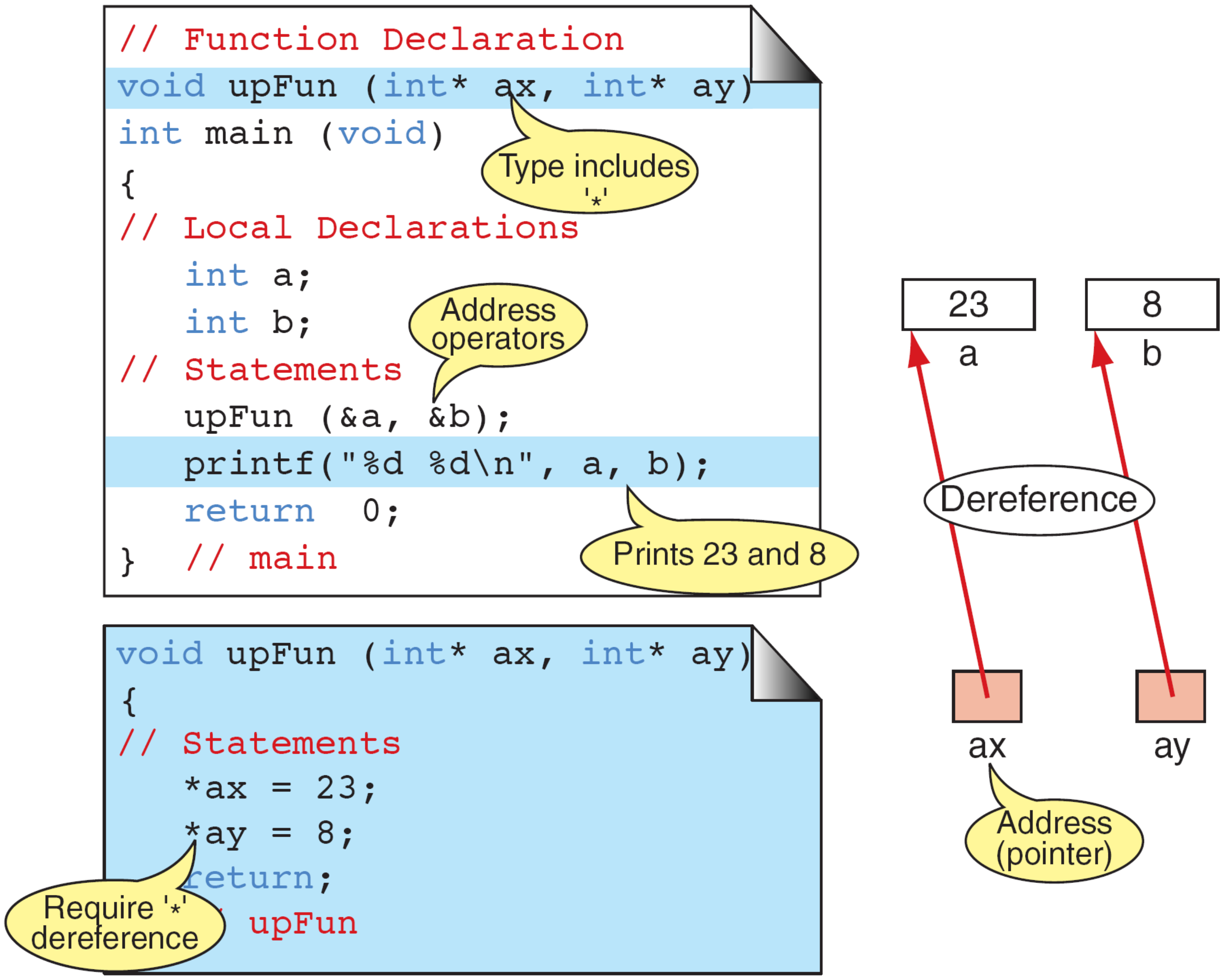
```
   void downFun (int x, int y)
   {
   // Statements
      x = x + y;
      return;
   }  // downFun
```

FIGURE 4-18 Downward Communication

```
int main (void)
{
   int a;
   int b;
   ...
   upFun (&a, &b);
   ...
}  // main
```

```
void upFun (int* ax, int* ay)
{
   *ax = 23;
   *ay = 8;
   return;
}  // upFun
```

FIGURE 4-19 **Upward Communication in C**



```
// Function Declaration
void upFun (int* ax, int* ay);
int main (void)
{
// Local Declarations
    int a;
    int b;
// Statements
    upFun (&a, &b);
    printf("%d %d\n", a, b);
    return  0;
}  // main
```

```
void upFun (int* ax, int* ay)
{
// Statements
    *ax = 23;
    *ay = 8;
    return;
     / upFun
```

FIGURE 4-20 Upward Communication

Note

To send data from the called function to the calling function:

1. **We need to use the & symbol in front of the data variable when we call the function.**
2. **We need to use the * symbol after the data type when we declare the address variable**
3. **We need to use the * in front of the variable when we store data indirectly**

```
int main (void)
{
   int a;
   int b;
   ...
   biFun (&a, &b);
   ...
}  // main
```

```
void biFun (int* ax, int* ay)
{
   *ax = *ax + 2;
   *ay = *ay / *ax;
   return;
}  // biFun
```

FIGURE 4-21 **Bi-directional Communication in C**

```
// Function Declaration
void biFun (int* ax, int* ay);

int main (void)
{
// Local Definitions
   int  a = 2;
   int  b = 6;

// Statements
   …
   biFun (&a, &b);
   …
 return 0;
}  // main
```

```
void biFun (int* ax, int* ay)
{
   *ax = *ax + 2;
   *ay = *ay / *ax;
   return;
}  // biFun
```

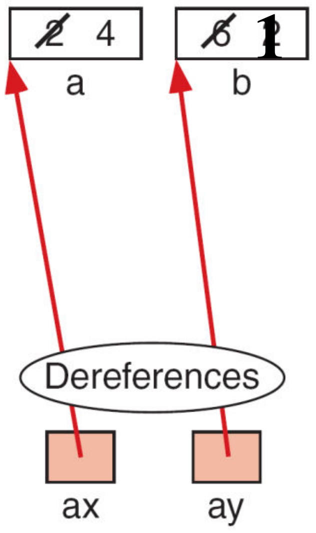
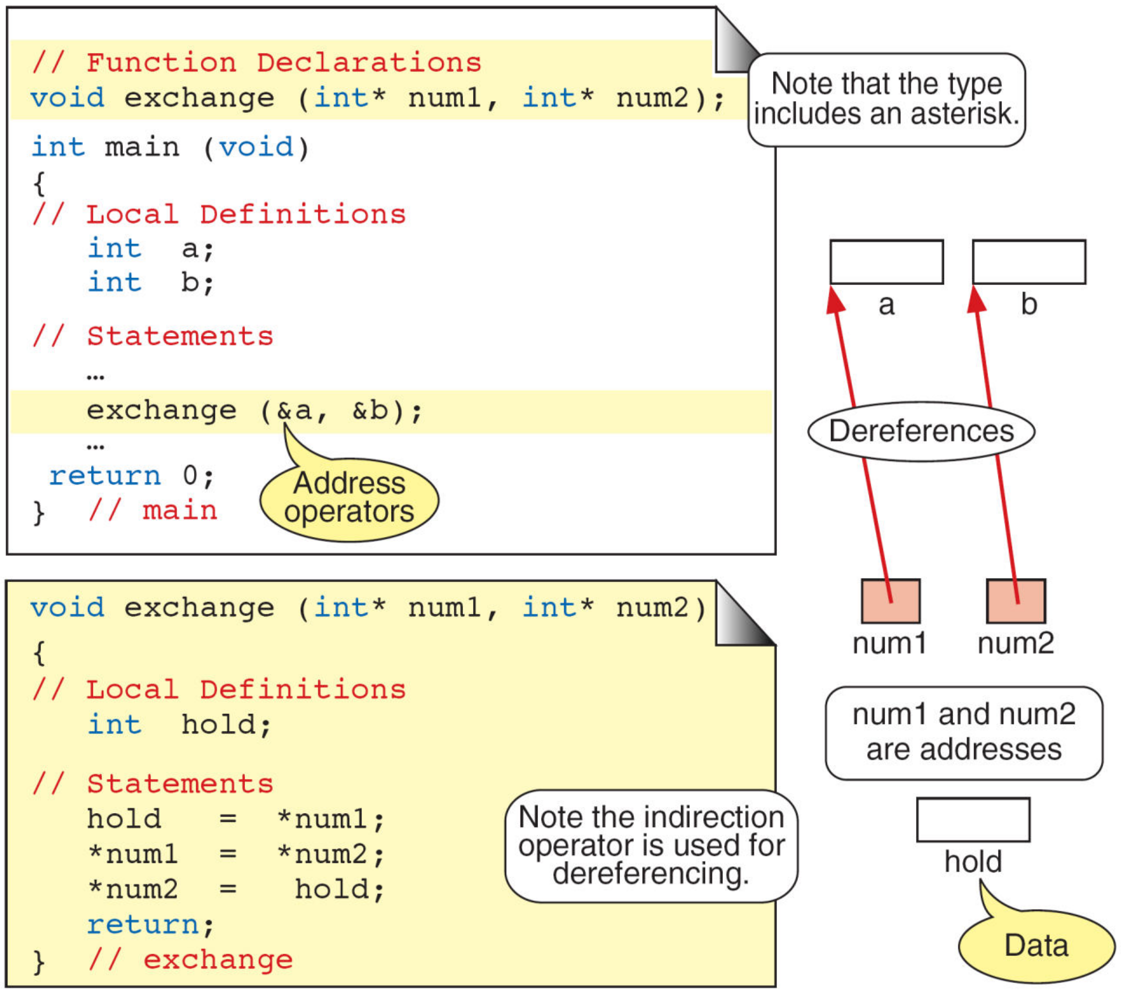


FIGURE 4-22 Bi-directional Communication



```
// Function Declarations
void exchange (int* num1, int* num2);
int main (void)
{
// Local Definitions
   int  a;
   int  b;

// Statements
   …
   exchange (&a, &b);
   …
 return 0;
}  // main
```

```
void exchange (int* num1, int* num2)
{
// Local Definitions
   int  hold;

// Statements
   hold   =  *num1;
   *num1  =  *num2;
   *num2  =   hold;
   return;
}  // exchange
```

Note that the type includes an asterisk.
Address operators
a
b
Dereferences
num1
num2
num1 and num2 are addresses
Note the indirection operator is used for dereferencing.
hold
Data

FIGURE 4-23 Exchange Function

```
// Function Declarations
void divide (int  divnd, int  divsr,
             int* quot,  int* rem);
int  main (void)
{
// Local Declarations
   int  a;
   int  b;
   int  quot;
   int  rem;
// Statements
    …
   divide (a, b, &quot, &rem);
    …
    return 0;
}  // main
```

```
void divide (int  divnd, int  divsr,
             int* quot,  int* rem)
// Statements
    {
   *quot =  divnd / divsr;
   *rem  =  divnd % divsr;
  return;
}  // divide
```

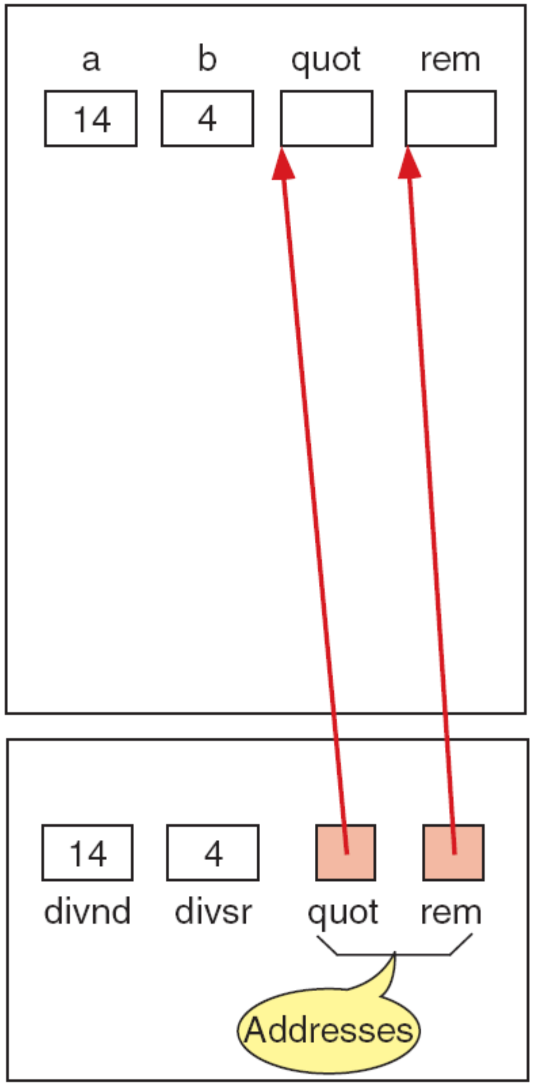


FIGURE 4-24 Calculate Quotient and Remainder

PROGRAM 4-8 Quotient and Remainder

```c
/* This program reads two integers and then prints the
   quotient and remainder of the first number divided
   by the second.
      Written by:
      Date:
*/
#include <stdio.h>

// Function Declarations
void divide  (int  dividend, int  divisor,
              int* quotient, int* remainder);
void getData (int* dividend, int* divisor);
void print   (int  quotient, int  remainder);

int main (void)
{
// Local Declarations
int  dividend;
```

PROGRAM 4-8 Quotient and Remainder

```c
int   divisor;
int   quot;
int   rem;

// Statements
   getData (&dividend, &divisor);
   divide  (dividend,   divisor, &quot, &rem);
   print   (quot, rem);

   return 0;
}  // main

/* ================== getData ==================
   This function reads two numbers into variables
   specified in the parameter list.
      Pre   Nothing.
      Post  Data read and placed in calling function.
*/
```

PROGRAM 4-8 Quotient and Remainder

```c
void getData  (int* dividend, int* divisor)
{
// Statements
   printf("Enter two integers and return: ");
   scanf ("%d%d", dividend, divisor);
   return;
}  // getData

/* ==================== divide =====================
   This function divides two integers and places the
   quotient/remainder in calling program variables
      Pre   dividend & divisor contain integer values
      Post  quotient & remainder calc'd
*/
void divide (int  dividend, int  divisor,
             int* quotient, int* remainder)
{
// Statements
   *quotient  = dividend / divisor;
```

PROGRAM 4-8 Quotient and Remainder

```c
   *remainder = dividend % divisor;
   return;
}  // divide

/* ==================== print ====================
   This function prints the quotient and the remainder
      Pre   quot contains the quotient
            rem contains the remainder
      Post  Quotient and remainder printed
*/
void print (int quot, int rem)
{
// Statements
   printf ("Quotient : %3d\n", quot);
   printf ("Remainder: %3d\n", rem);
   return;
}  // print
```