Pointers

Pointers

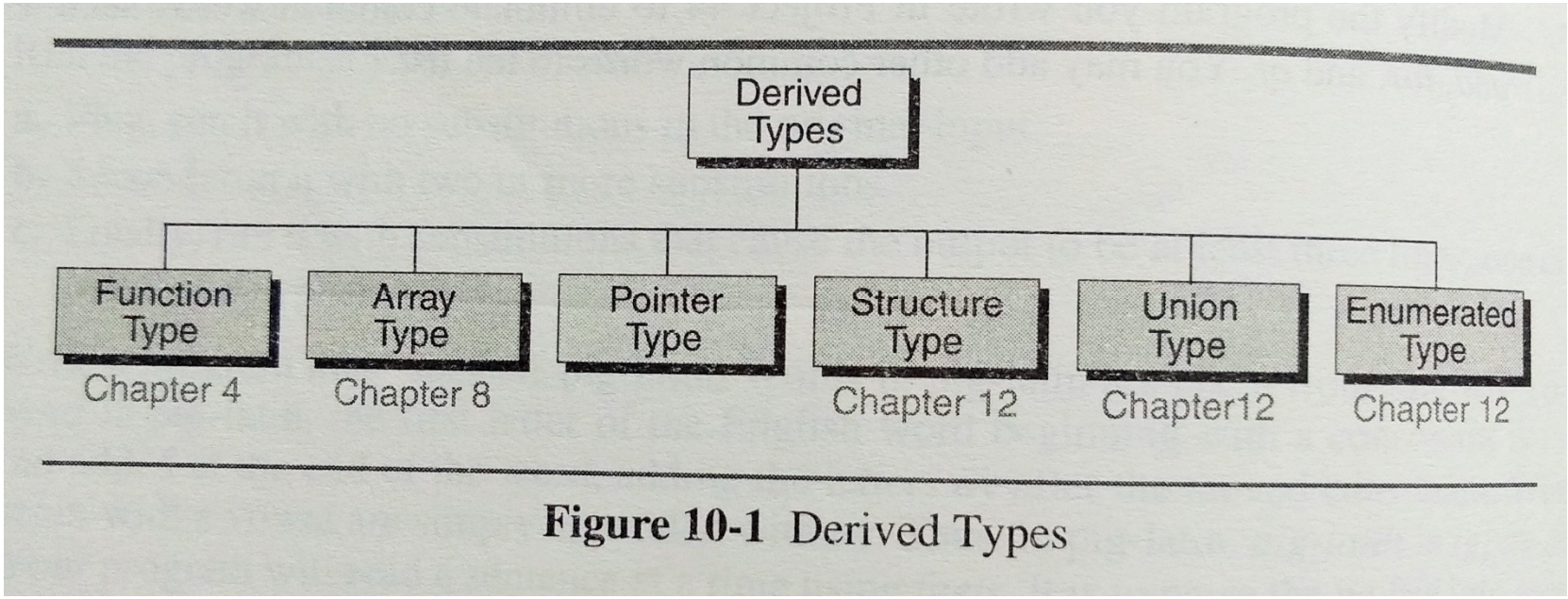


Figure 10-1 Derived Types

Introduction

- Pointer constants
- Pointer variables
- Accessing variables through pointers
- Pointer declaration and definition
- Initialization of pointer variables
- Pointers and functions
- Pointers to pointers

Pointers - Concepts

- Every computer has addressable memory locations
- Data Manipulation
- 1) Indirect Approach: Identifiers
- We use memory location addresses symbolically
  - We assign identifiers to data and then manipulate their contents through the identifiers
- 2) Direct Approach: Pointers
- Uses data addresses directly with ease and flexibility of symbolic names

Pointers

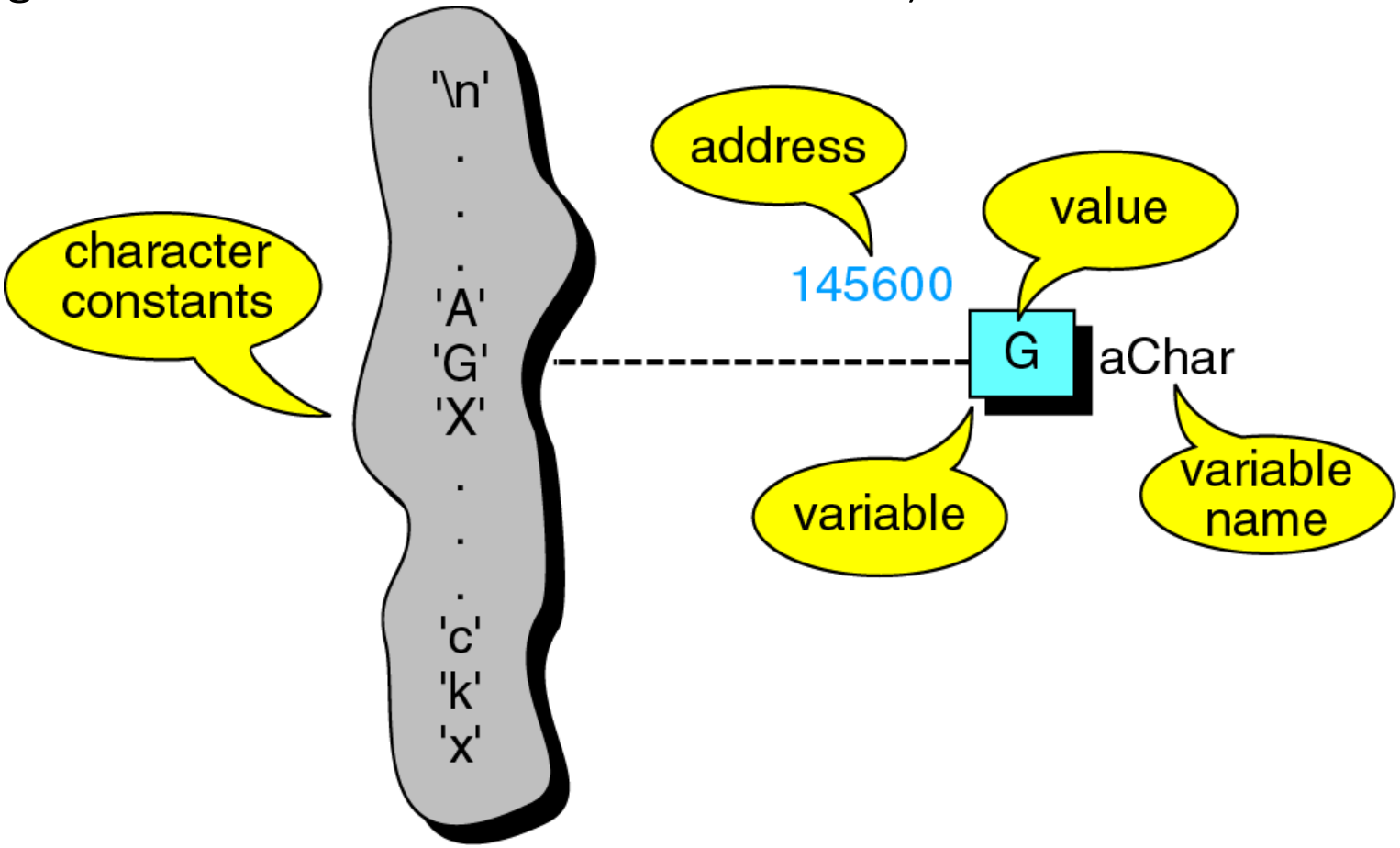
- A pointer is a derived data type : a data type built from one of the standard types
- Its value is any of the addresses available in the computer for storing and accessing data

Pointer Constants

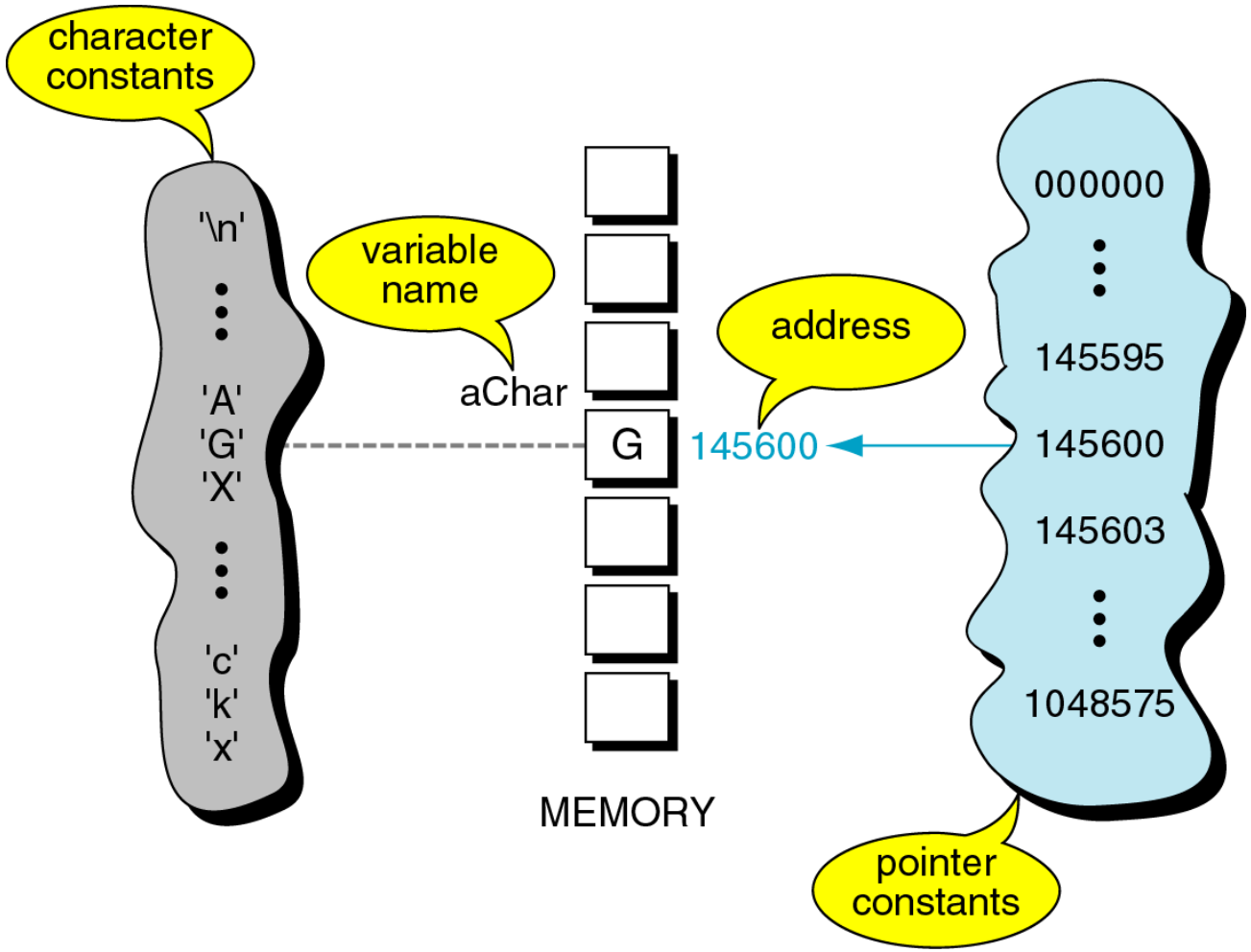
- First, compare character constants and pointer constants
- Character constant – we can have character constants from a universe of all characters
  - For most computers, it is known as ASCII
- A character constant can become a value and can be stored in a variable
- char aChar = ‘G’;

Pointer constants

(fig 10.2 Character constants & variables)



Pointer constants (fig 10.3)



Character Constants and Pointer Constants

- Like character constants, pointer constants cannot be changed
- The address for variable aChar is drawn from the set of pointer constants for our computer
- Although addresses within a computer cannot change (remains constant for the duration of the run)
  - but the address of a variable will change for each run of the program
  - Thus it is necessary to refer to pointer variables symbolically

Pointer values

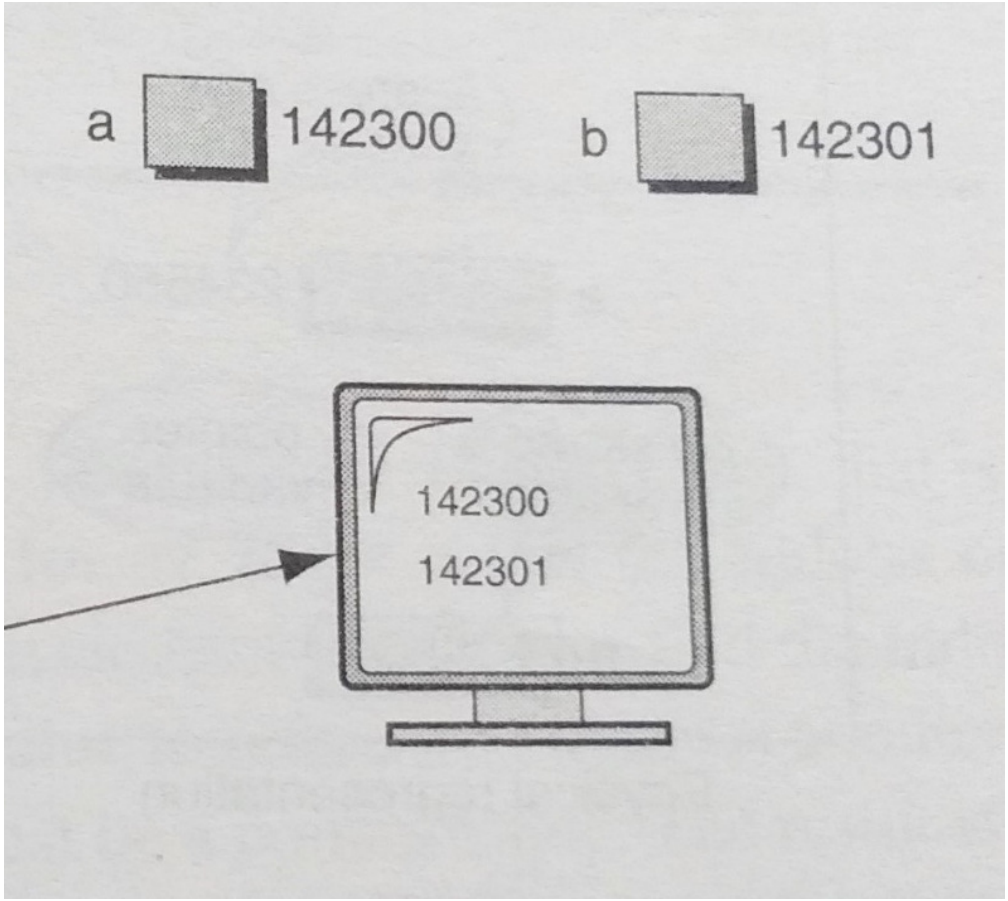
Pointer Constants

- An address in memory
- Drawn from the set of addresses for a computer
- Exist by themselves
- Cannot change them, only use them
- How to save this address?
- We have already done it by scanf with address operator &
- Ex:- WAP to print addresses as pointers

Pointer values (Fig 10.4 – Print Character Addresses)

```
int main (void)
{
// Local Declarations
   char a;
   char b;
// Statements
   printf ("%p\n %p\n", &a, &b);
   return 0;
}  // main
```

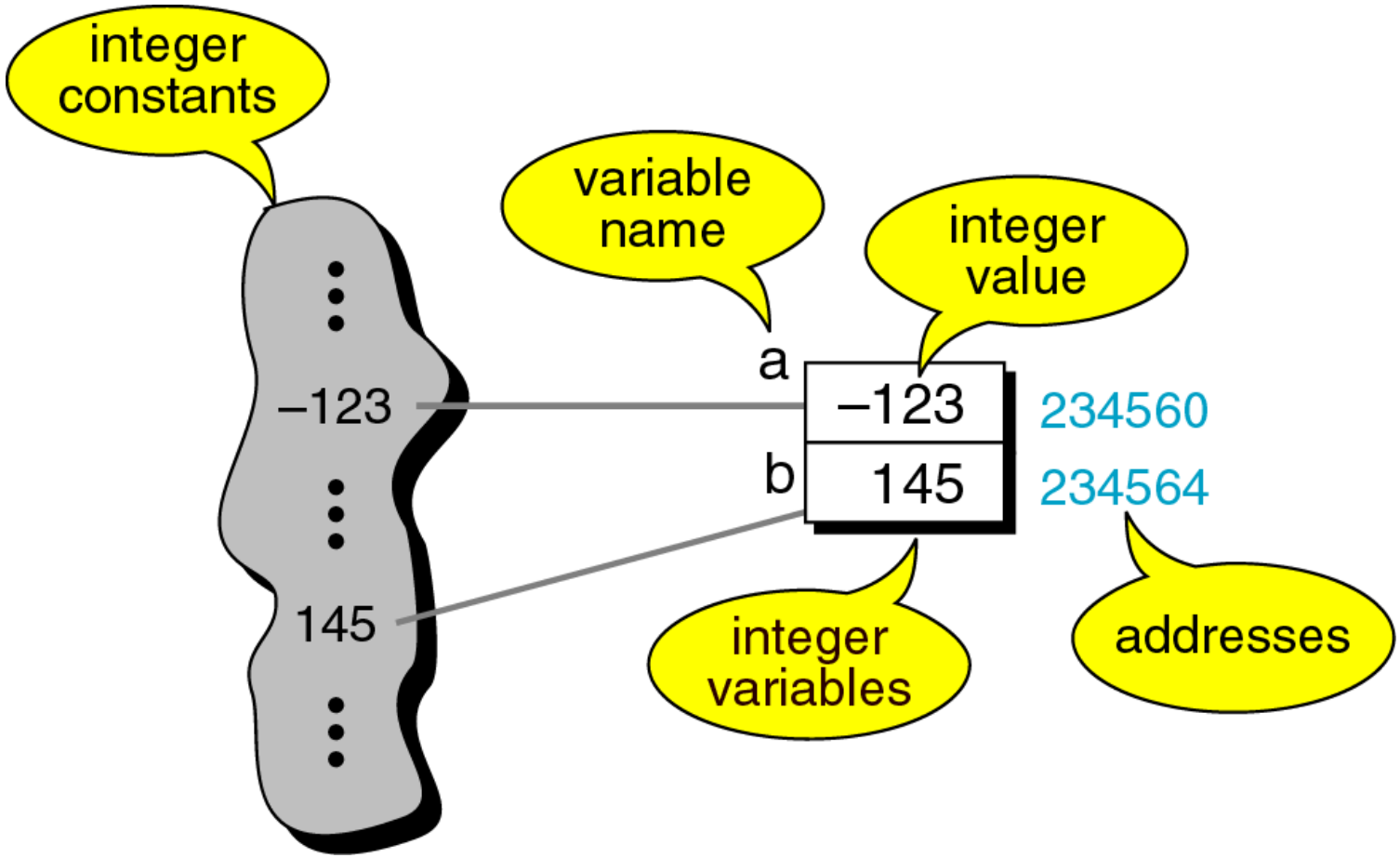
Pointer values (Fig 10.4 contd)



Integer Constants and variables

- In most computers, integers occupy either 2 or 4 bytes
- Assume we are working on a system with 4-byte integers
- This means that each integer occupies 4 memory locations
- Which one is then used to find the address of the variable?
  - The address of a variable is the address of the first byte occupied by that variable
  - For characters, there is only one byte, so its location is the address
  - For integers, the address is the first byte of the 4

Pointer values (fig 10.5 Integer constants & variables)



Pointer variables

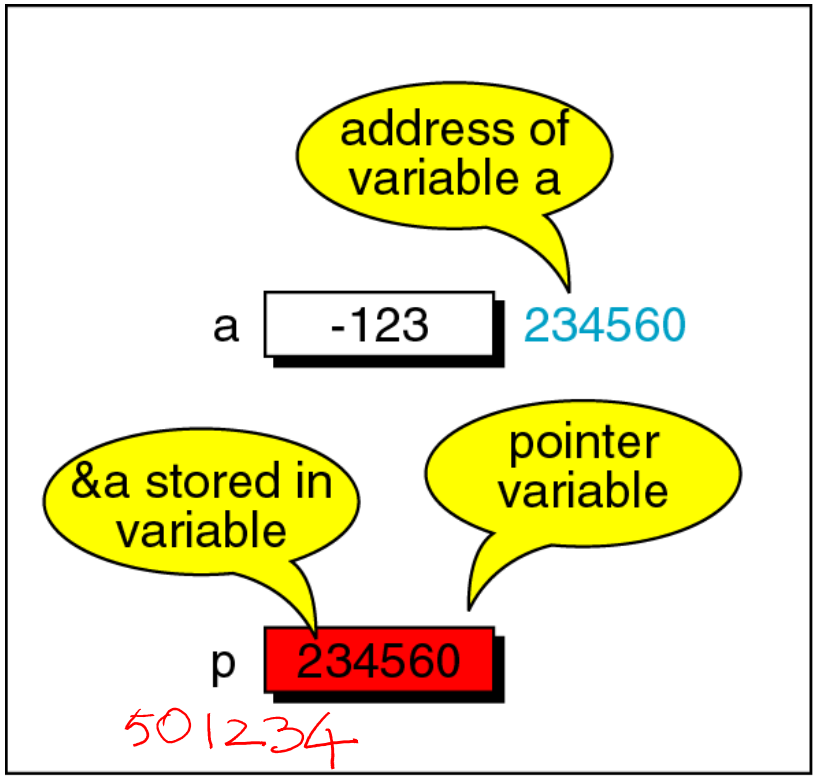
Pointer variables

- we have seen pointer constants and pointer values
- We can also have a pointer variable
- To store the address of a variable into another variable

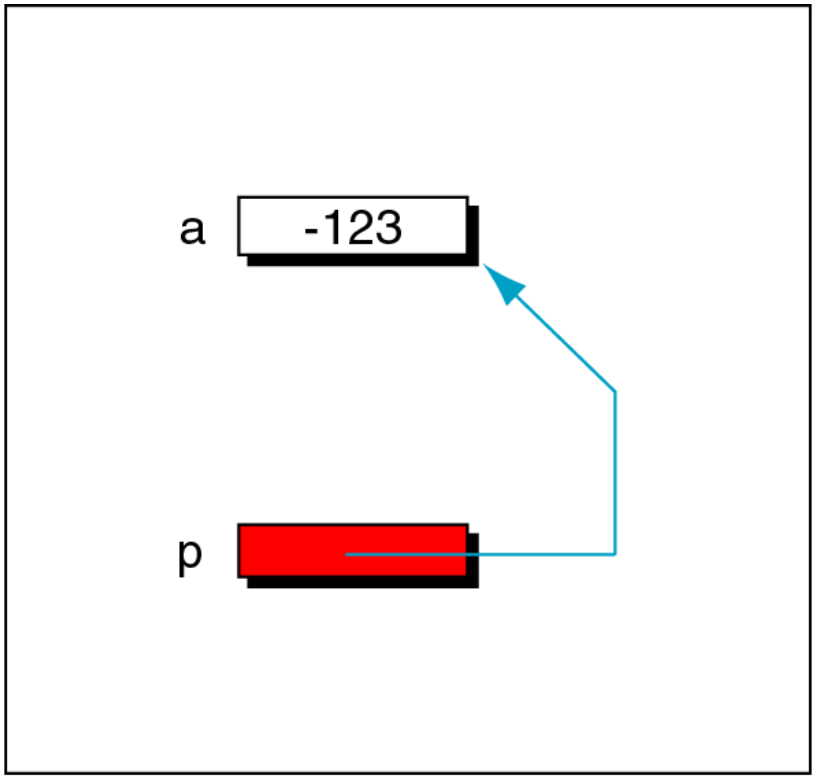
Pointer variables

- Distinguish between a pointer variable and its value
- There is an integer variable whose name and location are constant, the value may change as the program executes
- There is also a pointer which has a name and a location, both of which are constants

Pointer variables (fig 10.6)

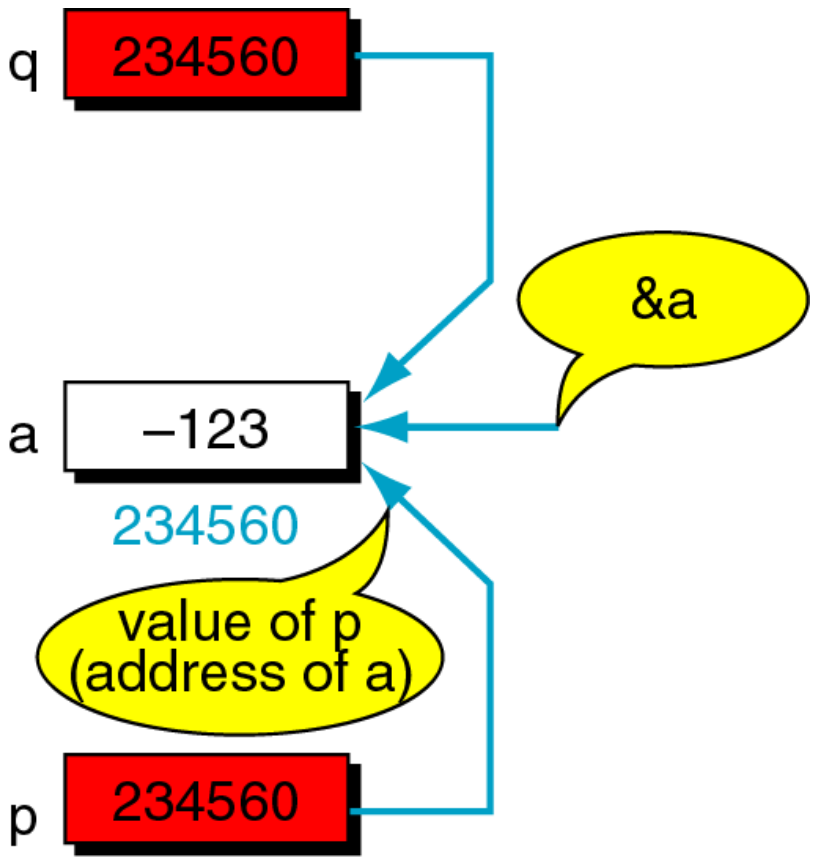


Physical representation



Logical representation

Pointer variables (fig 10.7 Multiple pointers to a variable)



Pointer variables

- What if we have a pointer variable, but do not want it to point anywhere?
- What is its value then?
- C provides a special null pointer constant NULL defined in standard input-output <stdio.h> library

```
int a;
int sum;
a=0;
sum=a+2;
```

Accessing variables through pointers

The indirection operator *

- We have a variable and a pointer to the variable
- How can we use the pointer?
- C has indirection operator *
- When we dereference a pointer, we are using its value to reference (address) another variable
- The indirection operator is a unary operator whose operand must be a pointer value
- To access the variable a, through the pointer p, we write *p

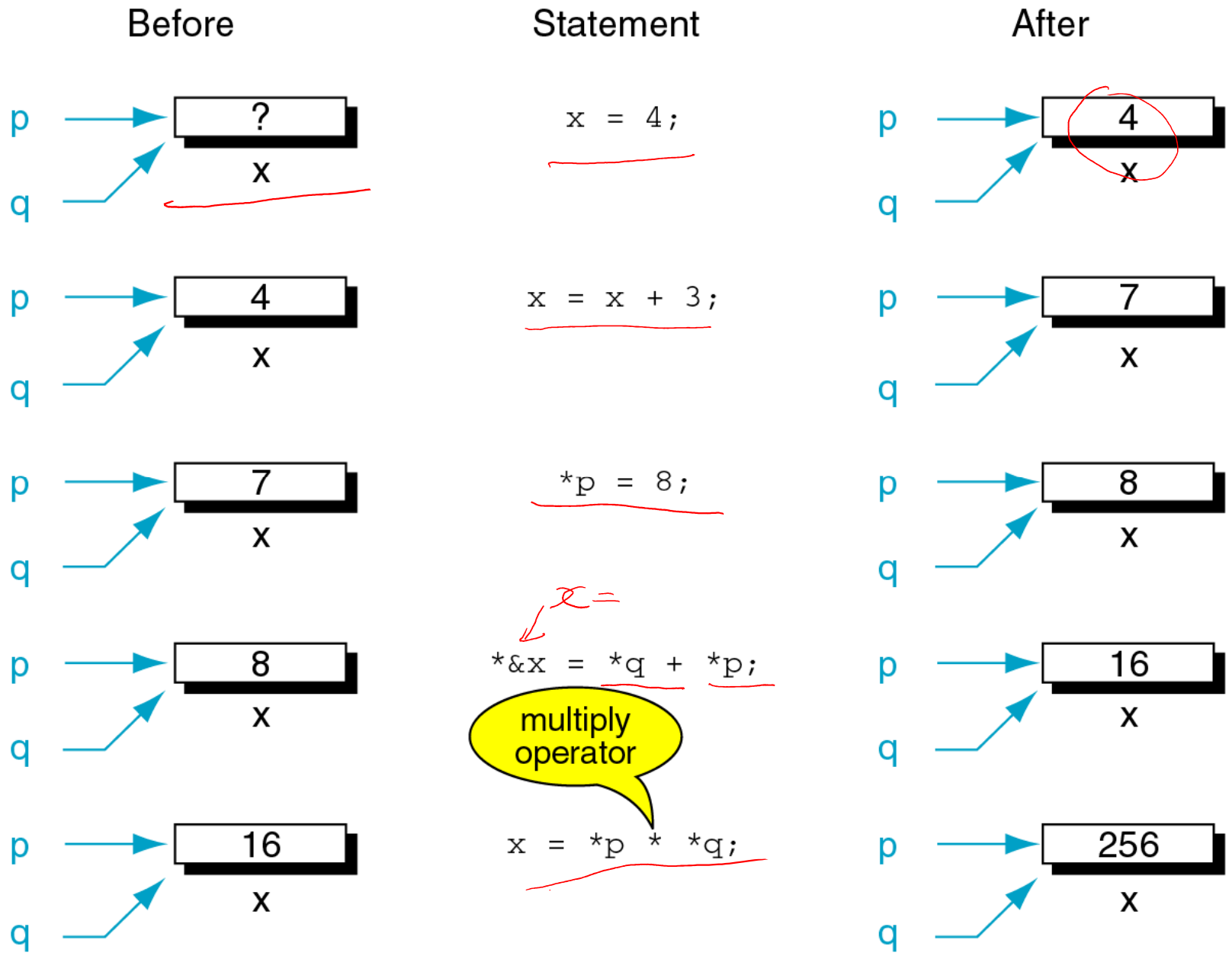
The indirection operator

- Add 1 to variable a using pointers and other means
- a++; a= a+1; *p = *p + 1; (*p)++;
- Any one of this statement will do it , assuming that the pointer p is properly initialized as p = &a;
- (*p)++ need parenthesis because ++ has more priority than *
- The parenthesis force dereference to occur and then addition
- Without parenthesis, we add to the pointer first, which would change the address

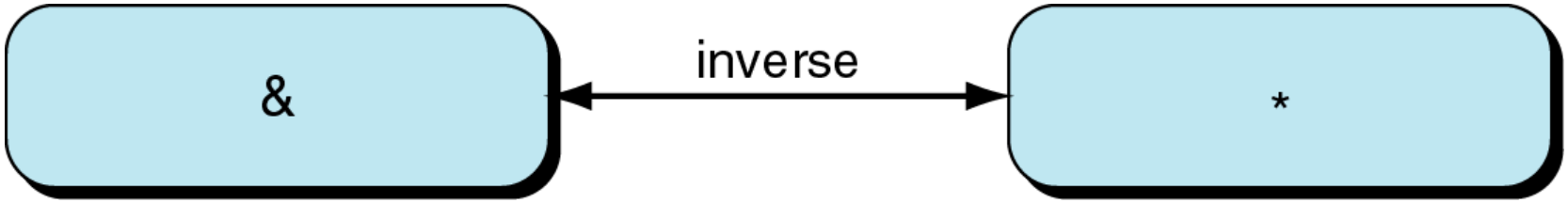
The indirection operator *

- Assume that the variable x is pointed to by two pointers p and q
- So x, *p, *q – allow the variable to inspect when used in RHS of the assignment operator
- When used in LHS, they change the value of x

Figure 10-8

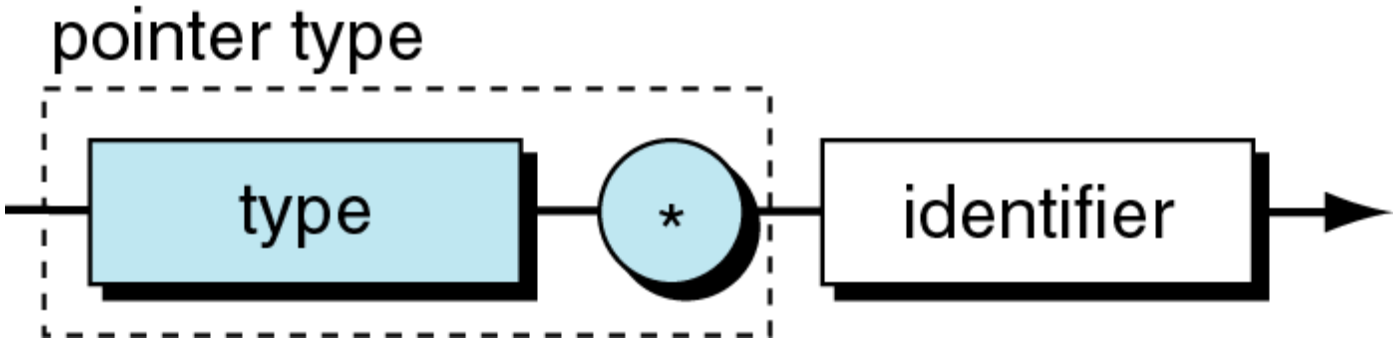

Before
Statement
After
p
q
?
x
x = 4;
4
x = x + 3;
7
*p = 8;
8
*&x = *q + *p;
16
multiply operator
x = *p * *q;
256

Accessing variables through pointers
(fig 10.9 Address & Indirection Operators)



Pointer declaration and definition

Figure 10-10 Pointer Variable Declaration



Pointer declaration & definition

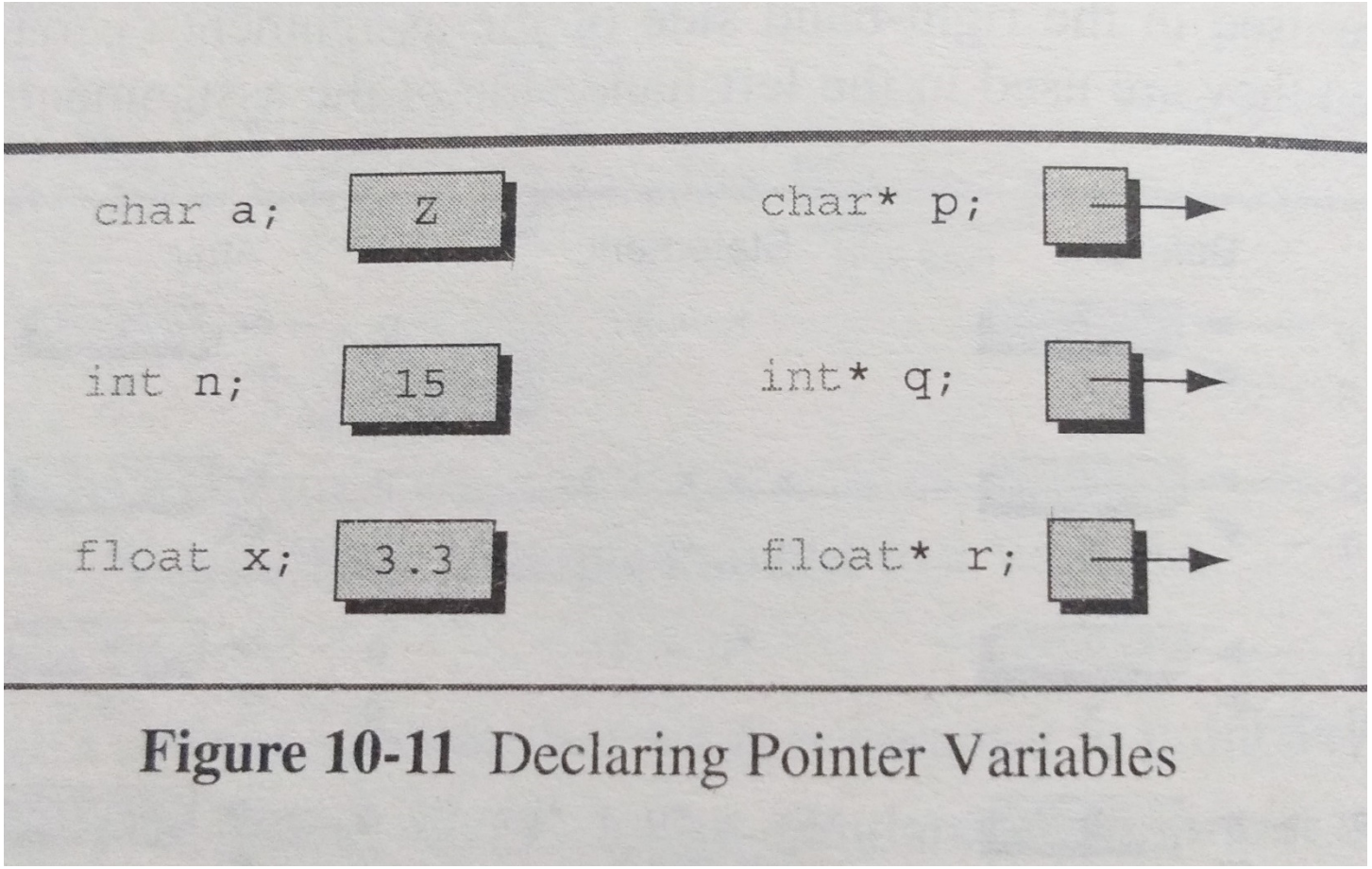


Figure 10-11 Declaring Pointer Variables

Problem

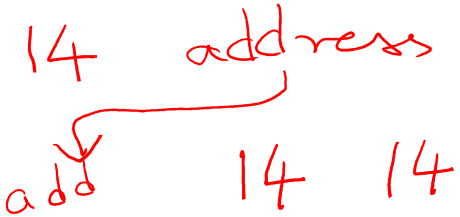
- WAP to define an integer variable a
- Define a pointer to integer and assign a’s address
- Print a and its address
- Print the pointer value containing the address of a

Pointer declaration & definition

(Prog 10.1 Demonstrate use of pointers)

```
int main (void) {
    int a;
    int *p;
    a = 14;
    p = &a;
    printf("%d  %p\n", a, &a);
    printf("%p  %d  %d\n", p, *p, a);
    return 0;
}
```

int* p;



Pointer declaration & definition
(Prog. 10.1 contd)

```
Results:
14 00135760
00135760 14 14
```

- Declaration versus Redirection

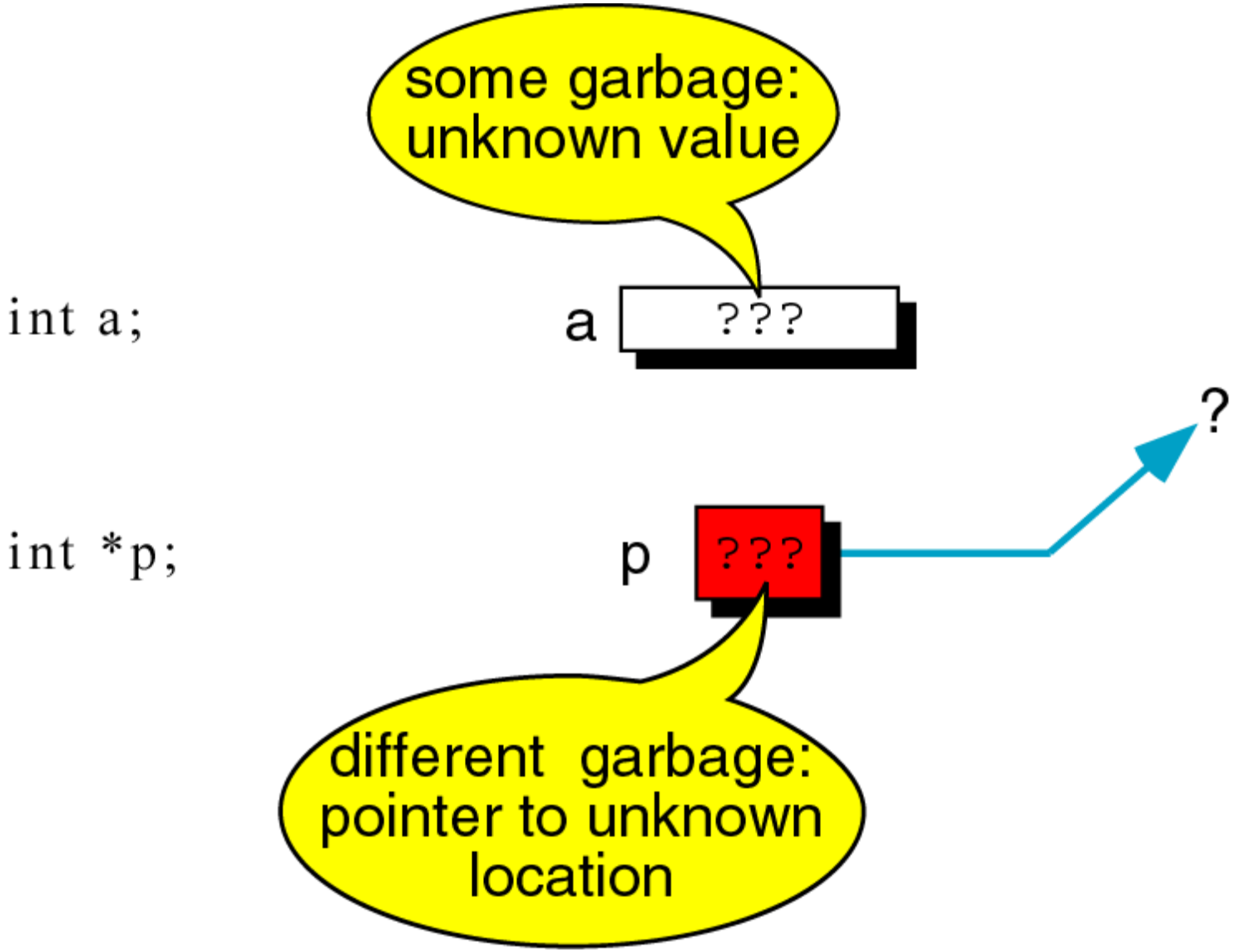
```
int *pa, *pb;
int sum;

sum = *pa + *pb;
```

Initialization of pointer variables

Initialization of pointer variables

- C language does not initialize variables
  - when we start our program uninitialized variables have garbage values

Figure 10-12 Uninitialized variables and Pointers



Initialization of pointer variables

- C language does not initialize pointer variables
- When the program starts, uninitialized pointer variables will have garbage address
- So assign a valid memory address to the pointer
- Ex:

```
int a;
int *p= &a;          // p has valid address
*p = 90;             // a is assigned 90
```

Initialization of Pointer Variables

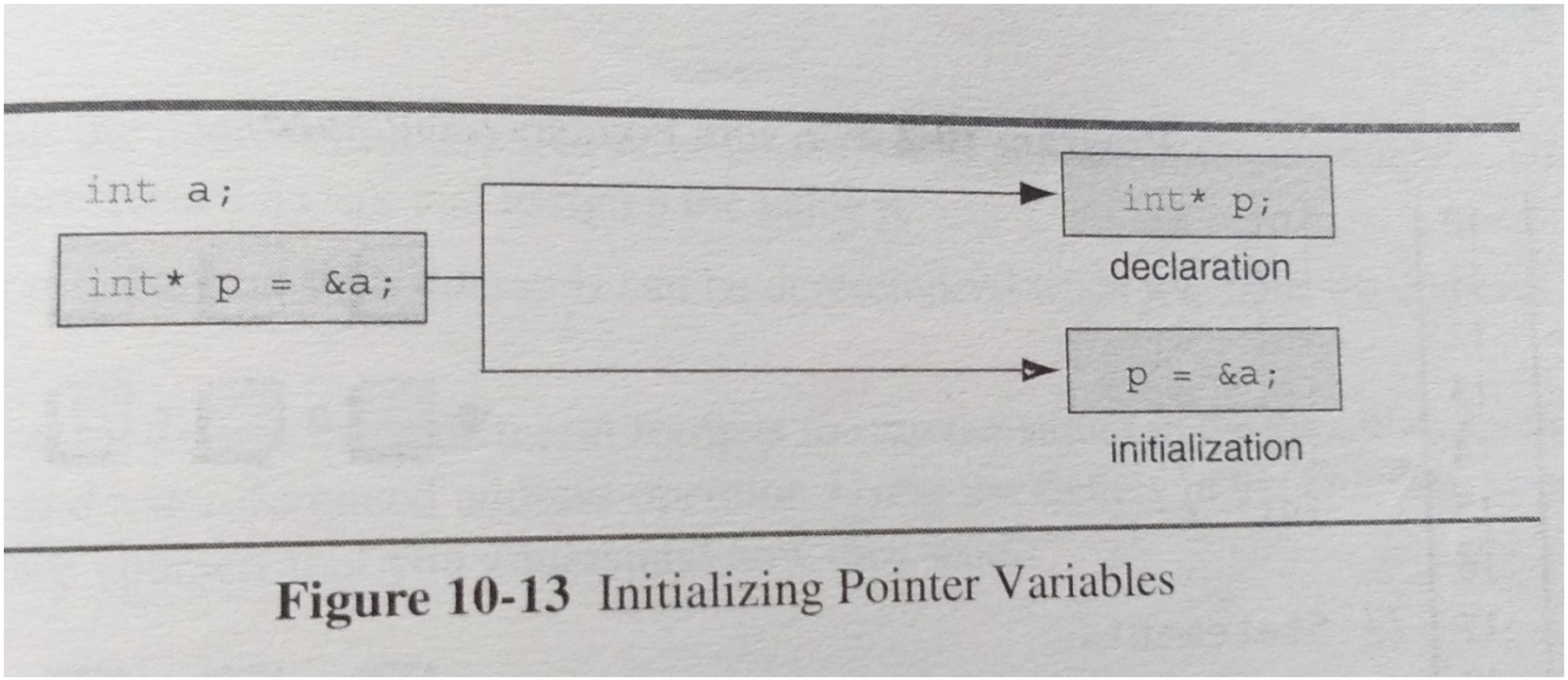


Figure 10-13 Initializing Pointer Variables

Initialization of pointer variables

- How to set pointer to null during definition or during execution?

  int *p = NULL;

- What happens when you dereference the pointer when it is null (or null pointer) ?
  - When we dereference a null pointer, we are using address zero
  - A valid address in the computer
  - Depending on OS, this can be the physical address zero or can be the first address location in our program area
  - In some systems
    - A Runtime error, NULL is not a valid address

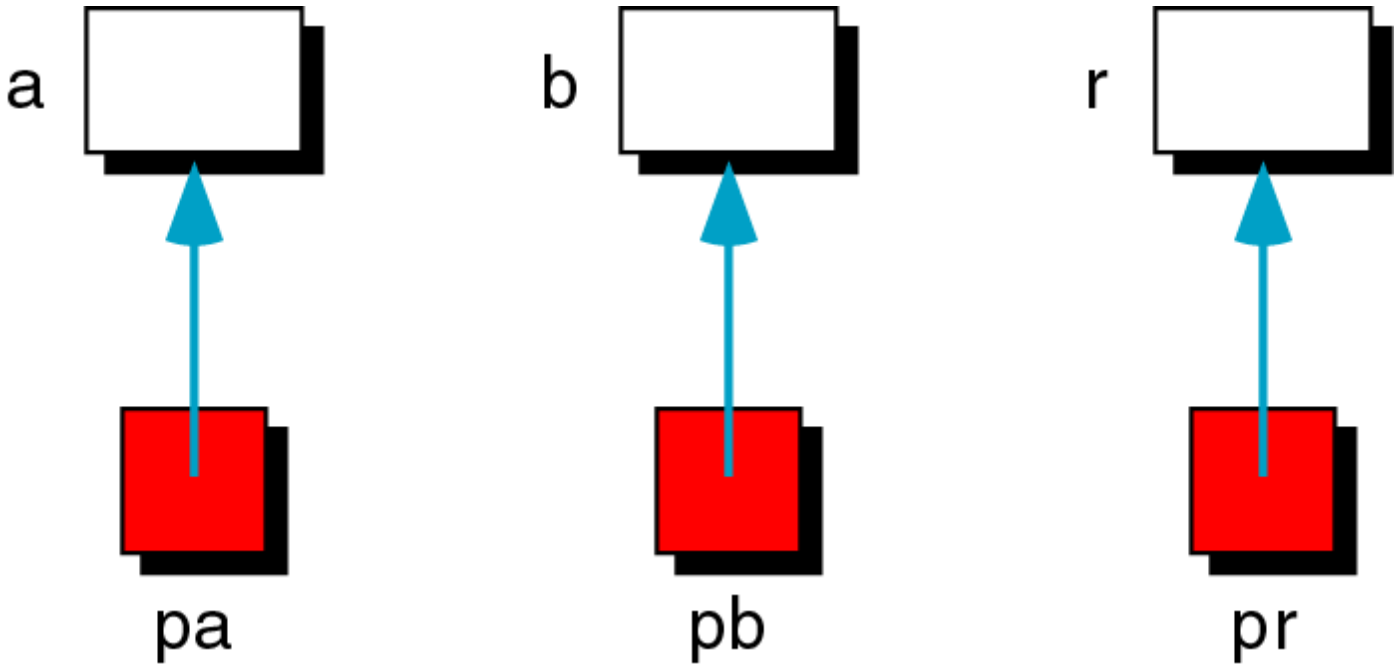
Change Variables (prog 10.2 contd)

```
int main (void) {
    int a, b, c;
    int *p, *q, *r;

    a = 6;
    b = 2;
    p = &b;
    q = p;
    r = &c;
    p = &a;
    *q = 8;
    *r = *p;
    *r = a + *q + *&c;

    printf("%d  %d  %d\n", a, b, c);
    printf("%d  %d  %d\n", *p, *q, *r);
    return 0;
}
```

Change Variables (prog 10.2 contd)

```
Results:
6 8 20
6 8 20
```

Figure 10-14 Add two numbers using pointers



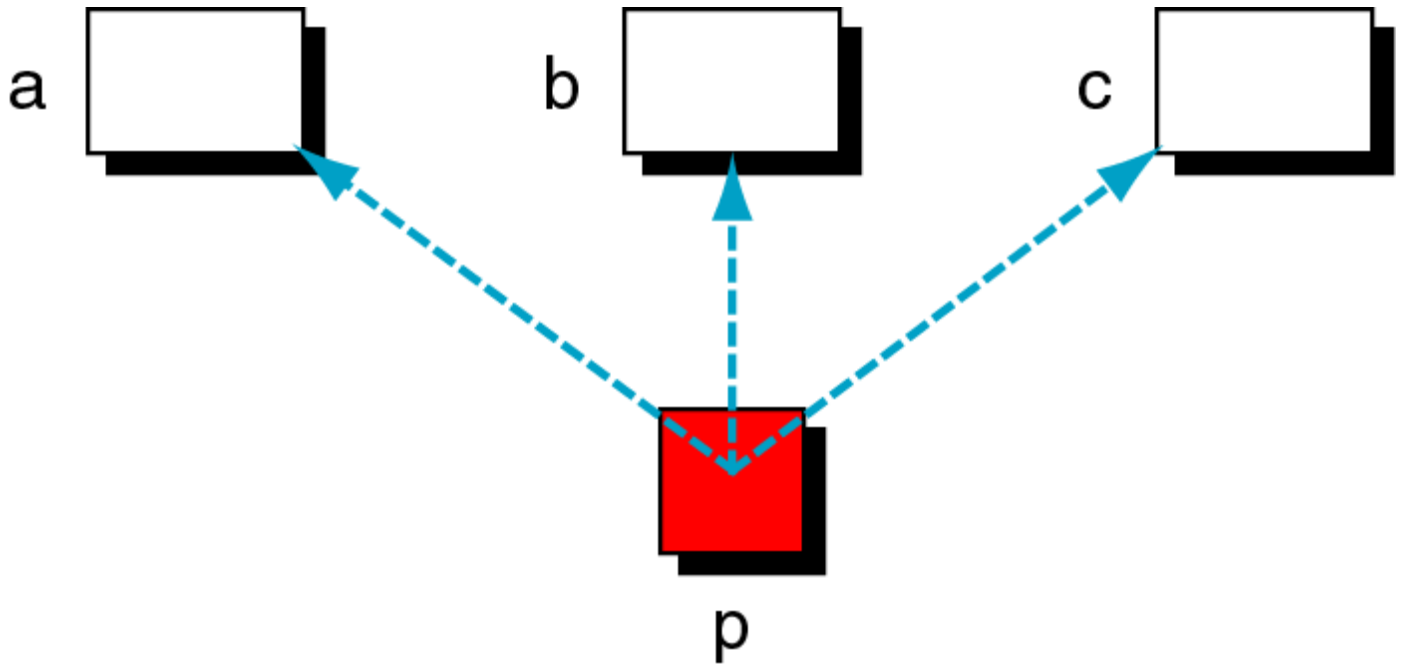
Add two numbers using pointers (prog 10.3)

```
int main (void) {
   int a, b, r;
   int *pa = &a;
   int *pb = &b;
   int *pr = &r;

   printf("Enter the first number : ");
   scanf("%d", &a);
   - - -
```

Add two numbers using pointers (prog 10.3)

```
int main (void) {
   int a, b, r;
   int *pa = &a;
   int *pb = &b;
   int *pr = &r;

   printf("Enter the first number : ");
   scanf("%d", pa);
   printf("Enter the second number : ");
   scanf("%d", pb);
   *pr = *pa + *pb;
   printf("\nThe sum is : %d\n", *pr);
   return 0;
}
```

Figure 10-15 Demonstrate pointer flexibility



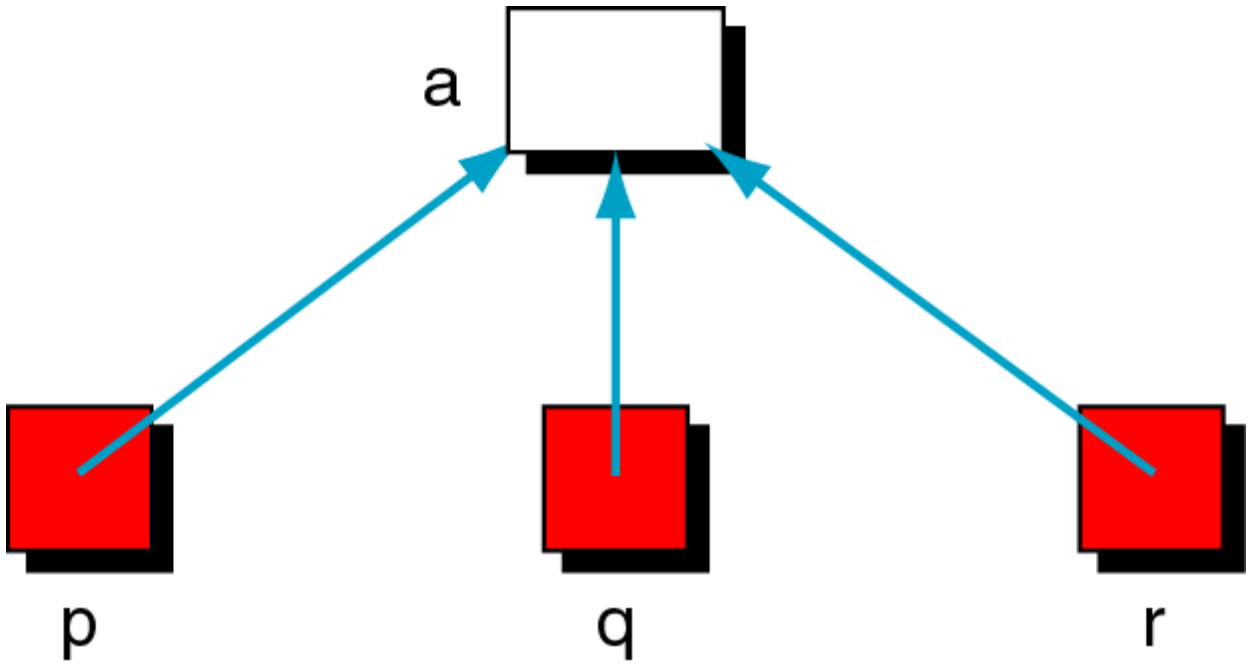
Using one pointer for many variables (prog 10.4)

```
int main (void) {
   int a, b, c;
   int *p;

   printf(“Enter three numbers : ”);
   scanf(“%d %d %d”, &a, &b, &c);

   - - - -
```

Using one pointer for many variables (prog 10.4)

```
int main (void) {
   int a, b, c;
   int *p;

   printf(“Enter three numbers : ”);
   scanf(“%d %d %d”, &a, &b, &c);
   p = &a;
   printf(“%d\n”, *p);
   p = &b;
   printf(“%d\n”, *p);
   p = &c;
   printf(“%d\n”, *p);
   return 0;
}
```

Figure 10-16 One variable with many pointers



Using a variable with many pointers (prog 10.5)

```
int main (void) {
   int a;
   int *p = &a;
   int *q = &a;
   int *r = &a;

   printf(“Enter a number : ”);
   scanf(“%d”, &a);
   printf(“%d\n”, *p);
   printf(“%d\n”, *q);
   printf(“%d\n”, *r);
   return 0;
}
```

Pointers and Functions

Pointers for Inter-function communication

Passing addresses (fig 10.17)

```
void exchange (int x, int y);

int main (void)
{
   int a = 5;
   int b = 7;
   exchange (a, b);
   printf("%d %d\n", a, b);
   return 0;
} // main
```

Pointers for Inter-function communication

Passing addresses (fig 10.17 contd)

```
void exchange (int x, int y)
{
   int temp;

   temp = x;
   x    = y;
   y    = temp;
   return;
} // exchange
```

Pointers and Functions

- Functions receiving pointer values as arguments
  - Ex1: exchange program without pointers
  - Ex2: exchange program with pointers

Pointers and Functions

- Most useful application of pointers is in functions
- How does C function operate?
  - C uses pass by value concept
  - This means that the only direct way to send something back from a function is through return value
- How to simulate pass by address?
  - We can simulate the pass by reference by passing an address and using it to refer back to data in the calling program
  - When we pass by address, we are actually passing a pointer to a variable

Pointers and Functions

- pass pointers to the values
- Given a pointer to a variable anywhere in our memory space
  - whether it is local to a function or main() or a global variable
  - we can change the content of the variable

Pointers and Functions

- It is important to understand that C still uses pass by value
  - Now the value is the address of the variable we need to change
  - We are only simulating pass by reference

Pointers and Functions

- If we want a called function to have access to a variable in the calling function send the address of that variable to the called function and use the indirection operator * to access it
- To send back more than one value from a function, use pointers
- By passing the address of variables defined in calling function, we can store data directly in the calling function rather than using return

Pointers for Inter-function communication

Passing addresses (fig 10.18 the correct way)

```
void exchange (int*, int*);

int main (void)
{
   int a = 5;
   int b = 7;

   exchange (&a, &b);
   printf("%d %d\n", a, b);
   return 0;
} // main
```

Pointers for Inter-function communication

Passing addresses (fig 10.18 the correct way - contd)

```
void exchange (int* px, int* py)
{
   int temp;

   temp = *px;
   *px  = *py;
   *py  = temp;
   return;
} // exchange
```

Functions Returning Pointers

Pointers and Functions

- Functions returning a pointer to the calling function
  - Ex: program to find the smaller of two numbers by taking address of two numbers as input and returning address of smaller number as output

Pointers for Inter-function communication

Functions returning pointers (fig 10.19)

```
int* smaller (int* p1, int* p2);

int main (void)
  …
  int  a;
  int  b;
  int* p;
  …
  scanf ( "%d %d", &a, &b );
  p = smaller (&a, &b);
  …
```

Pointers for Inter-function communication

Functions returning pointers (fig 10.19 contd)

```
int* smaller (int* px, int* py)
{
  return (*px < *py ? px : py);
} // smaller
```

Functions Returning Pointers – Points to note

- It is a serious error to return a pointer to a local variable
- When we return a pointer, it must point to data in the calling function or a higher level function
- It is an error to return a pointer to a local variable in the called function because
  - When the function terminates, its memory may be used by other parts of the program
  - Especially evident in large programs

Pointers to Pointers

Pointers to Pointers

- All pointers have been pointing directly to data
- Advanced data structures require
- Pointers that point to other pointers
- Ex: we can have a pointer pointing to a pointer to an integer
- This two-level indirection is shown next

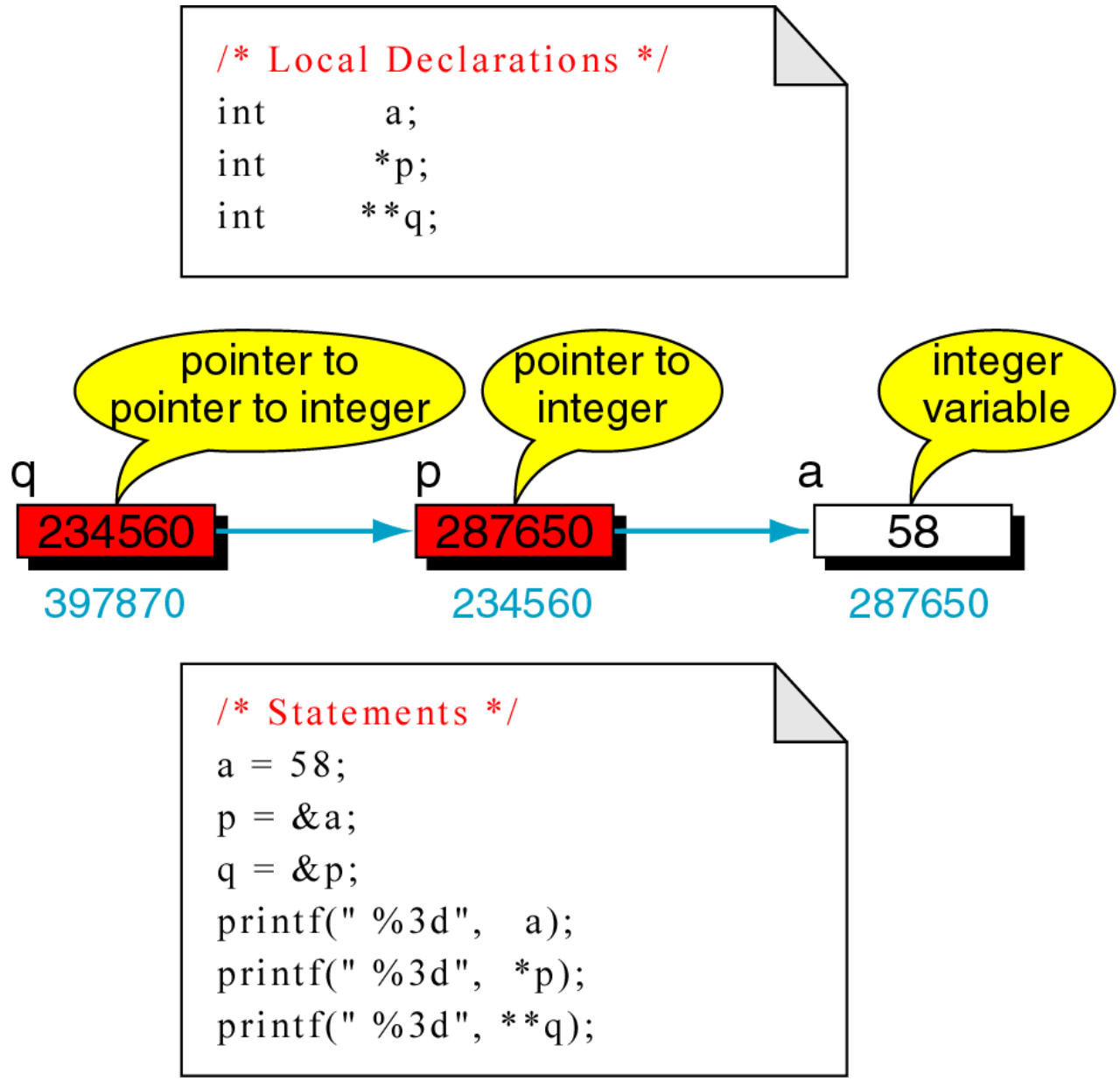
Note: Although many levels of indirection can be used but practically not more than two levels are needed

Pointers to Pointers (fig 10.20)

```
// Local Declarations
int   a;
int*  p;
int** q;
```

```
// Statements
a = 58;
p = &a;
q = &p;
printf(" %3d",   a);
printf(" %3d",  *p);
printf(" %3d", **q);
```

Figure 10-20 Pointers to pointers



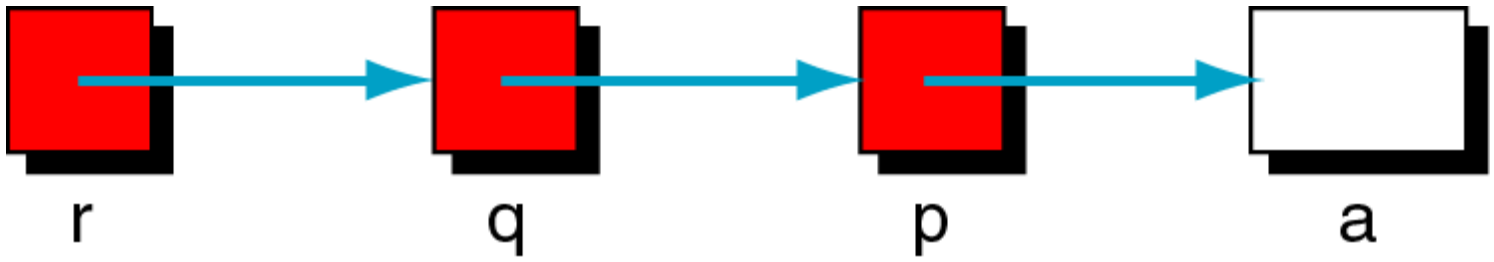
Pointers to Pointers

- Each level of pointer indirection requires a separate indirection operator when it is dereferenced
- Ex: To refer to **a** using the pointer p, we have to dereference it once as
- *p
- To refer to a using the pointer q, we need to dereference it twice to get to the integer a as there are 2 levels of indirection (pointers) involved
- By first dereference, we reference p, which is a pointer to an integer

Pointers to Pointers

- q is a pointer to a pointer to an integer
- **q
- Program:
- WAP to print the value of a by 4 means:
  - Directly using a
  - Using pointer p
  - Using pointer q
  - Using pointer r

Figure 10-21 Using Pointers to Pointers



Pointers to Pointers (fig 10.6)

```
int main (void)
{
// Local Declarations
   int    a;
   int*   p;
   int**  q;
   int*** r;

// Statements
   p = &a;
   q = &p;
   r = &q;
```

Pointers to Pointers (fig 10.6 contd)

```
printf("Enter a number: ");
scanf ("%d", &a);
printf("The number is : %d\n", a);

printf("\nEnter a number: ");
scanf ("%d", p);
printf("The number is : %d\n", a);

printf("\nEnter a number: ");
scanf ("%d", *q);
printf("The number is : %d\n", a);

printf("\nEnter a number: ");
scanf ("%d", **r);
printf("The number is : %d\n", a);
```

End