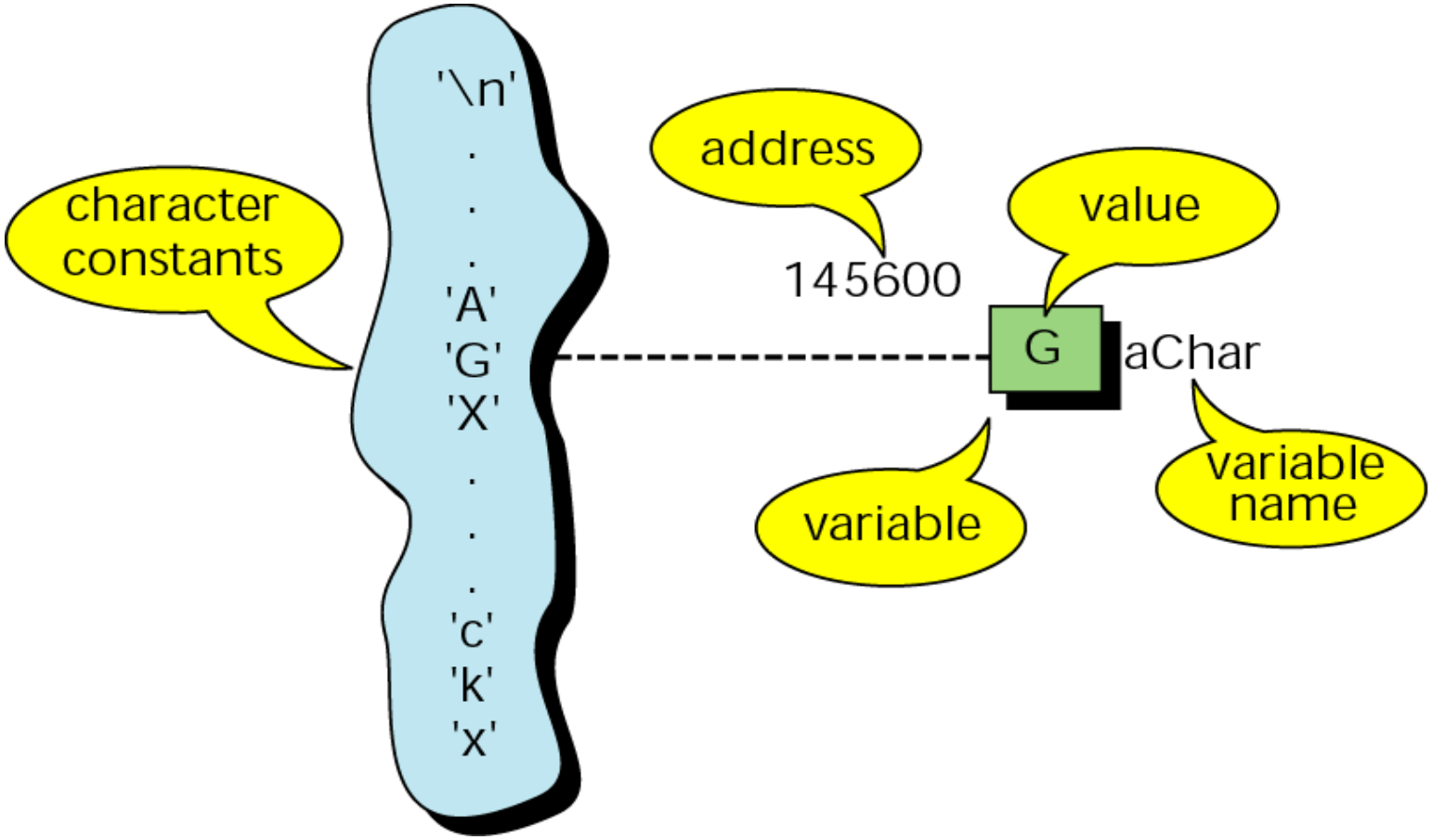
UNIVERSITY of LOUISVILLE
dare to be great

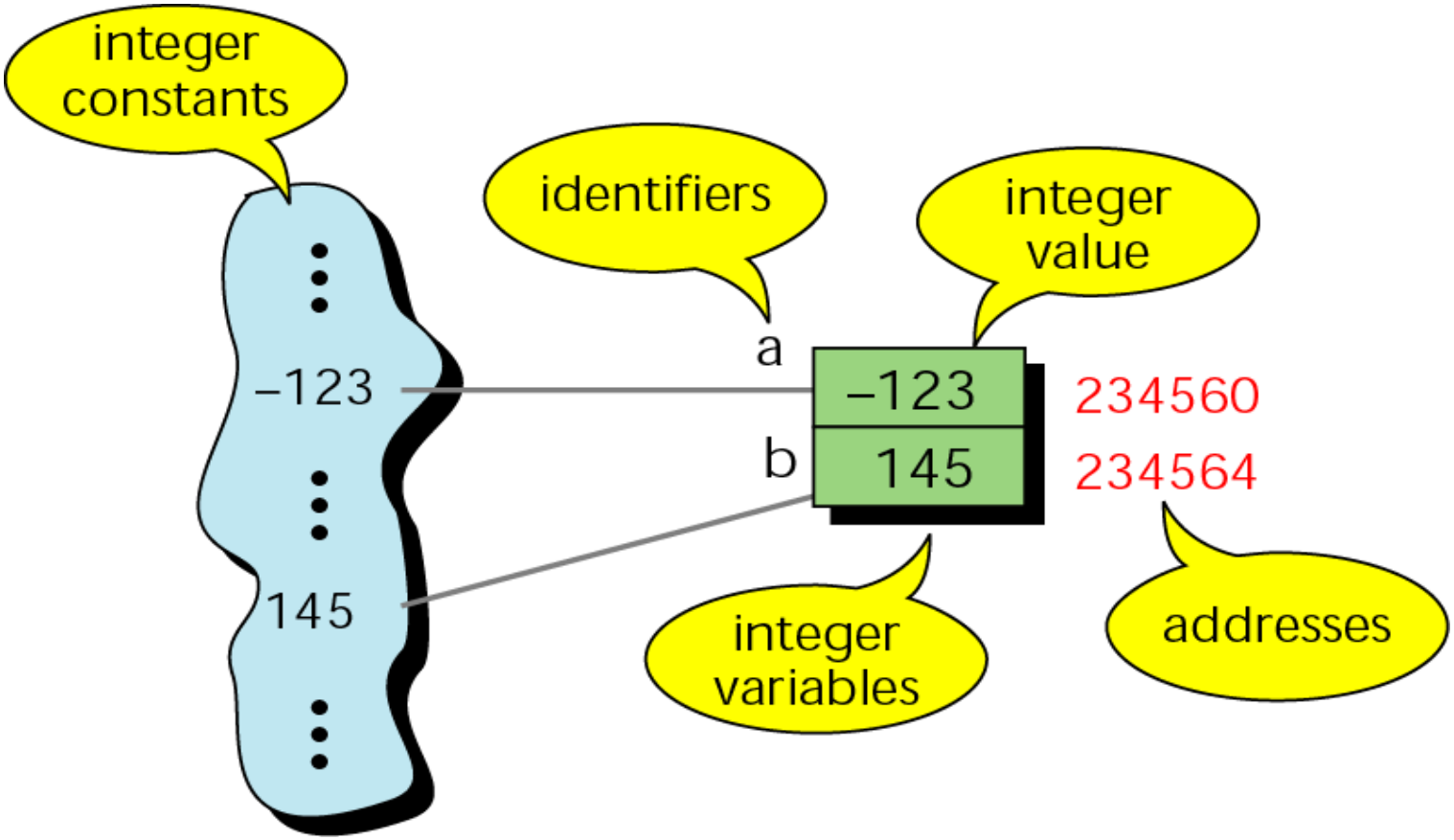
C Chapter 7: Pointers

CECS130
Introduction to Programming Languages
Dr. Roman V. Yampolskiy

Character Constants and Variables



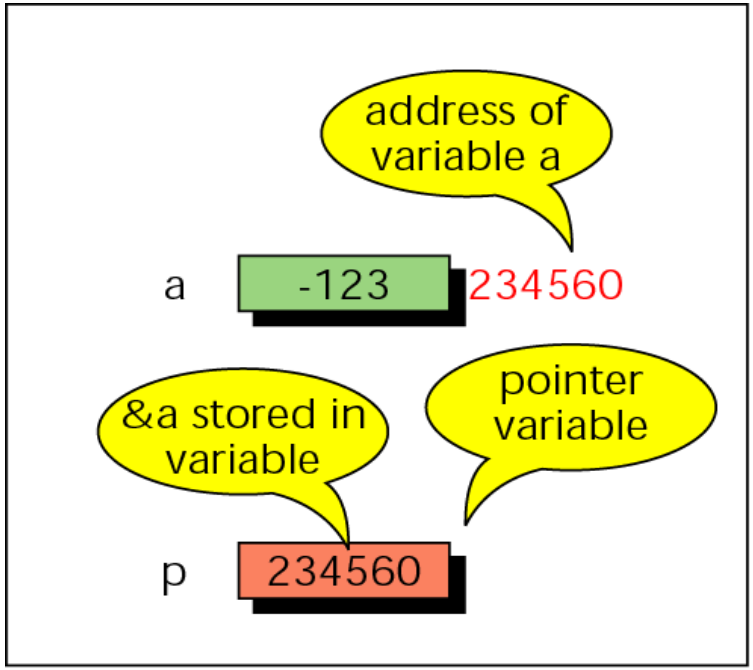
Integer Constants and Variables



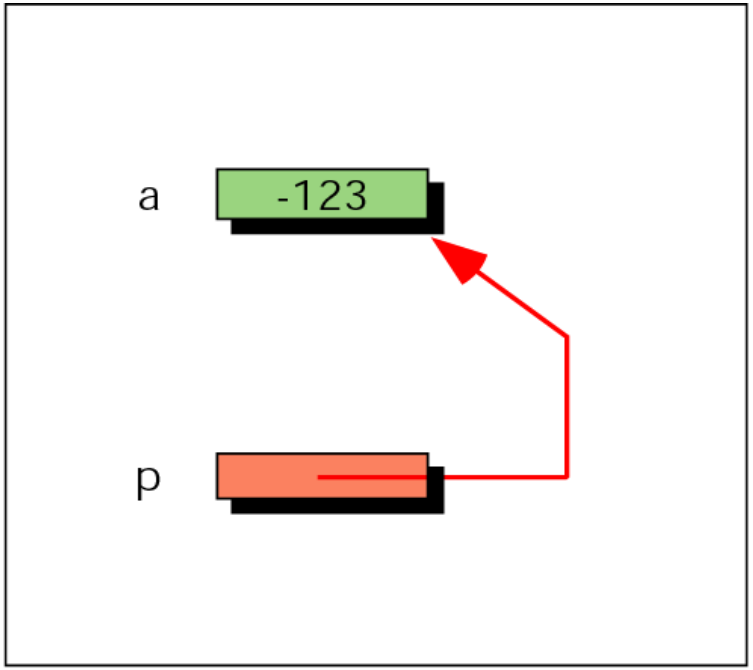
What is a Pointer?

- *Pointer variables*, simply called *pointers*, are designed to hold memory addresses as their values.
- Normally, a variable contains a specific value, e.g., an integer, a floating-point value, or a character.
- However, a pointer contains the memory address of a variable that in turn contains a specific value.

Pointer Variable

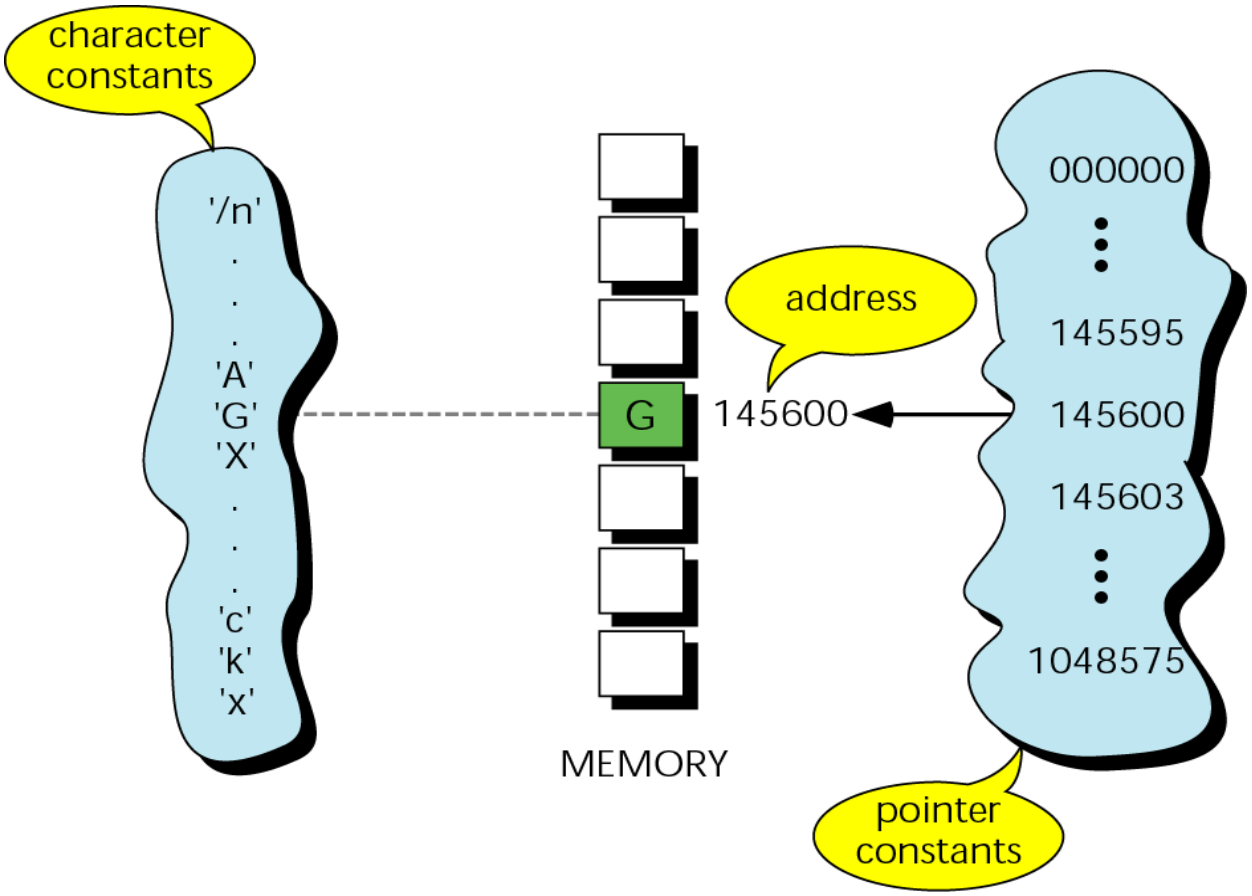


Physical representation



Logical representation

Pointer Constants



Note:

Pointer constants, drawn from the set of addresses for a computer, exist by themselves. We cannot change them; we can only use them.

Declare a Pointer

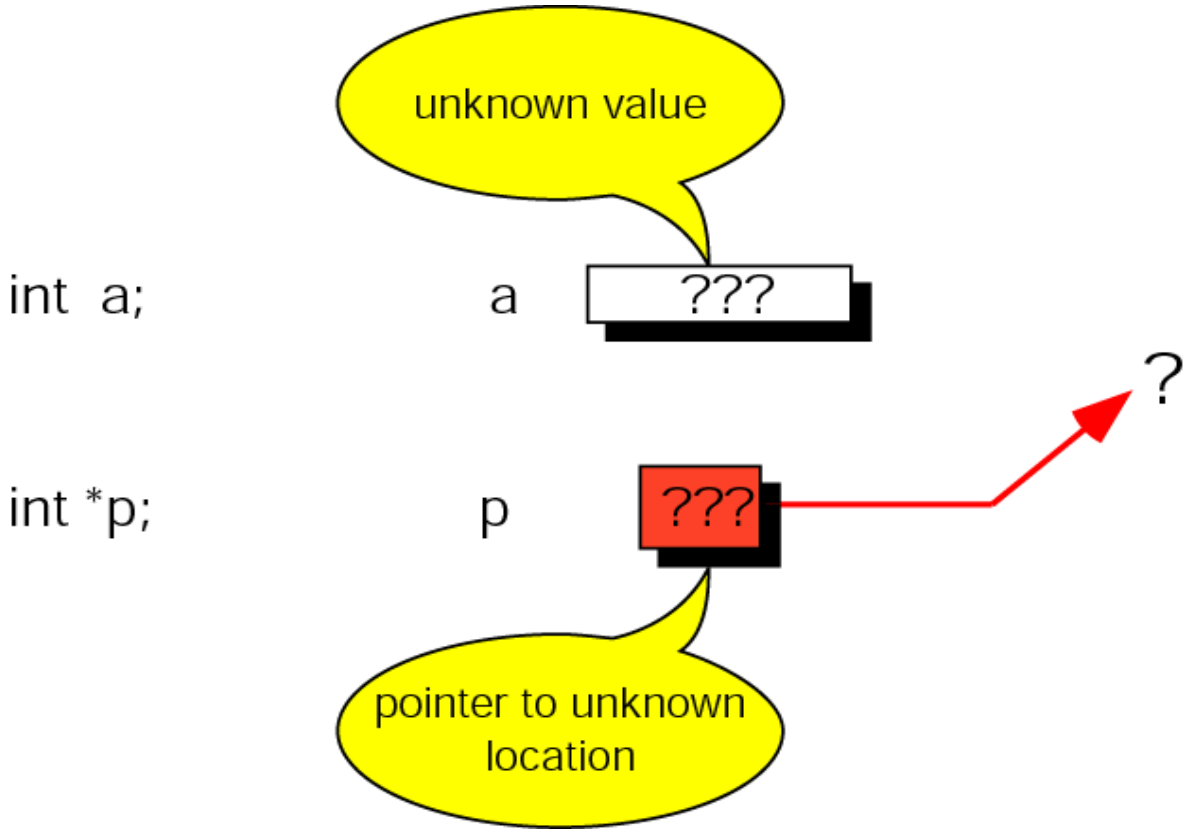
Like any other variables, pointers must be declared before they can be used. To declare a pointer, use the following syntax:

dataType *pVarName;

Each variable being declared as a pointer must be preceded by an asterisk (*). For example, the following statement declares a pointer variable named pCount that can point to an int varaible.

int *pCount;

Uninitialized Pointers



Initializing Pointer

Like a variable, a pointer is assigned an arbitrary value if you don't initialize it.

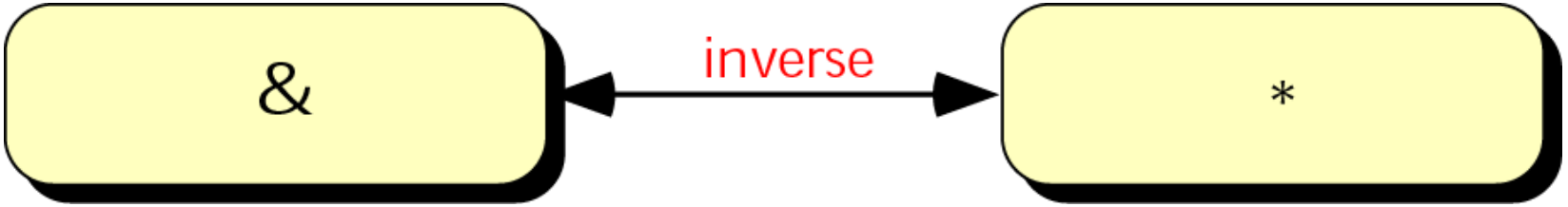
A pointer may be initialized to 0 or NULL, which are special values to indicate that the pointer points to nothing. [int *ptrGrade = NULL;]

You should always initialize pointers to prevent errors.

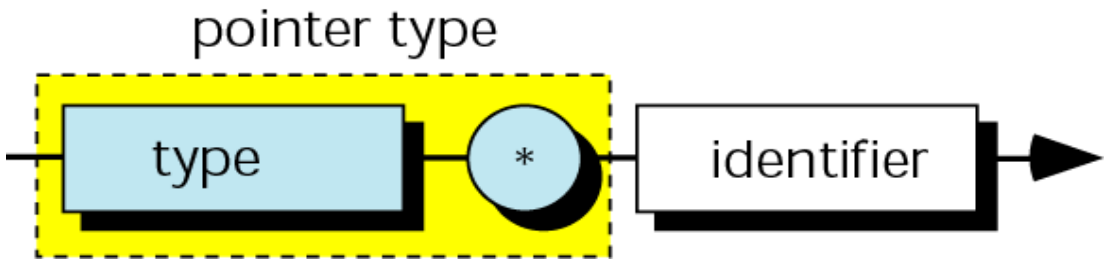
Dereferencing a pointer that is not initialized could cause fatal runtime error or it could accidentally modify important data.

& and *

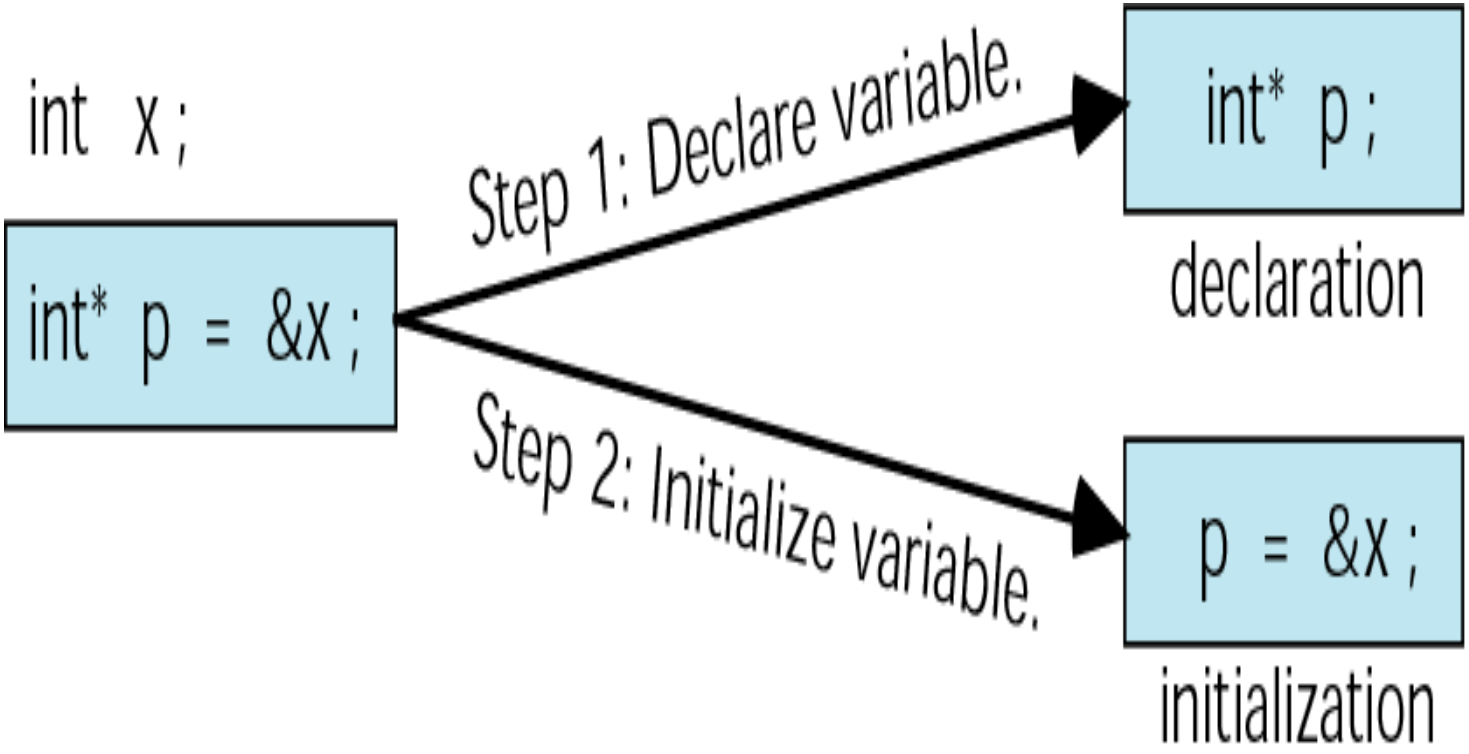
Address and indirection operators



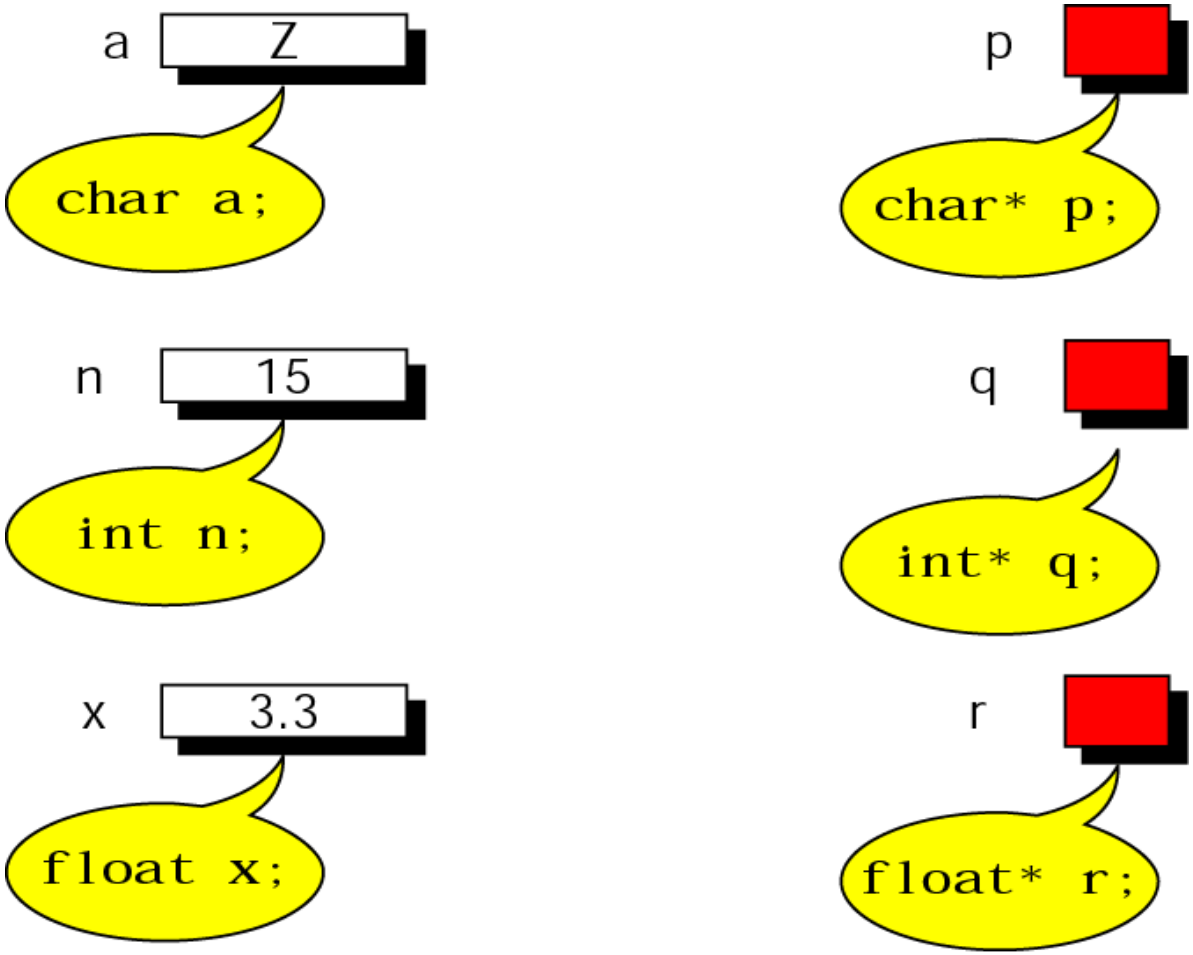
Pointer variable declaration



Initializing Pointer Variables



Declaring Pointer Variables



Dereferencing

•Referencing a value through a pointer is called *indirection*. The syntax for referencing a value from a pointer is:

*pointer

•For example, you can increase count using:

```
count++; // direct reference
```

•or

```
(*pCount)++; // indirect reference
```

Pointer Type

A pointer variable is declared with a type such as int, double, etc.
You have to assign the address of the variable of the same type.
It is a syntax error if the type of the variable does not match the type of the pointer.
For example, the following code is wrong.

```
int area = 1;
double *pArea = &area; // Wrong
int *pArea2 = &area; // Right
```

Note:

When the ampersand (&) is used as a prefix to a variable name, it means "address" of variable.

Print Character Address

```
#include <stdio.h>

int main() {

  char someChar = 'x';
  printf(“%p\n", &someChar);

  system("pause");
  return 0;
}
```

Output: 0022FF47

Note:

The address of a variable is the address of the first byte occupied by that variable.

Caution with Declarations

You can declare two variables on the same line. For example, the following line declares two int variables:

```
int i = 0, j = 1;
```

Can you declare two pointer variables on the same line as follows?

```
int* pI, pJ;
```

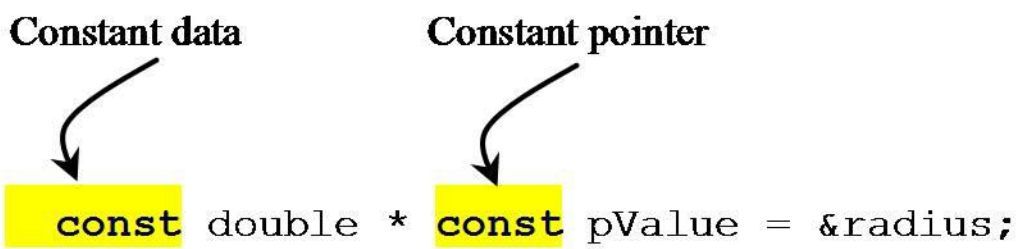
No, this line is equivalent to

```
int *pI, pJ;
```

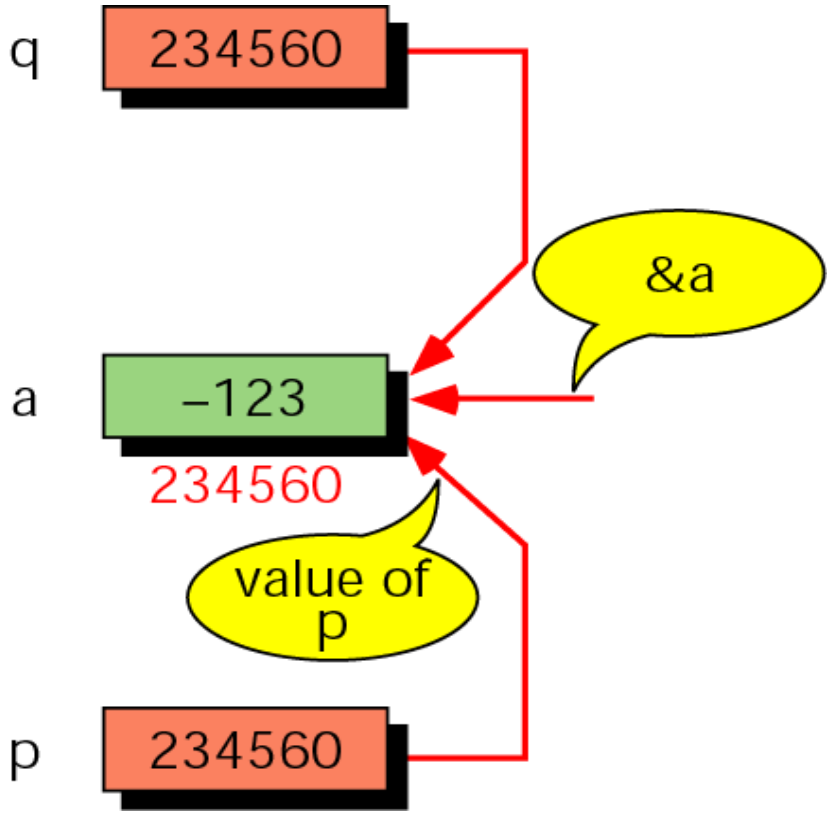
Using const with Pointers

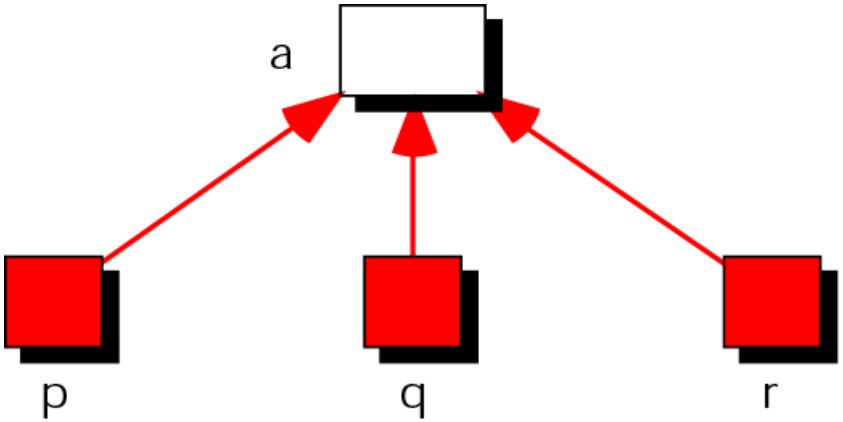
You learned how to declare a constant using the const keyword.
A constant cannot be changed once it is declared.
You can declare a constant pointer.
For example, see the following code:

```
double radius = 5;
double * const pValue = &radius;
```

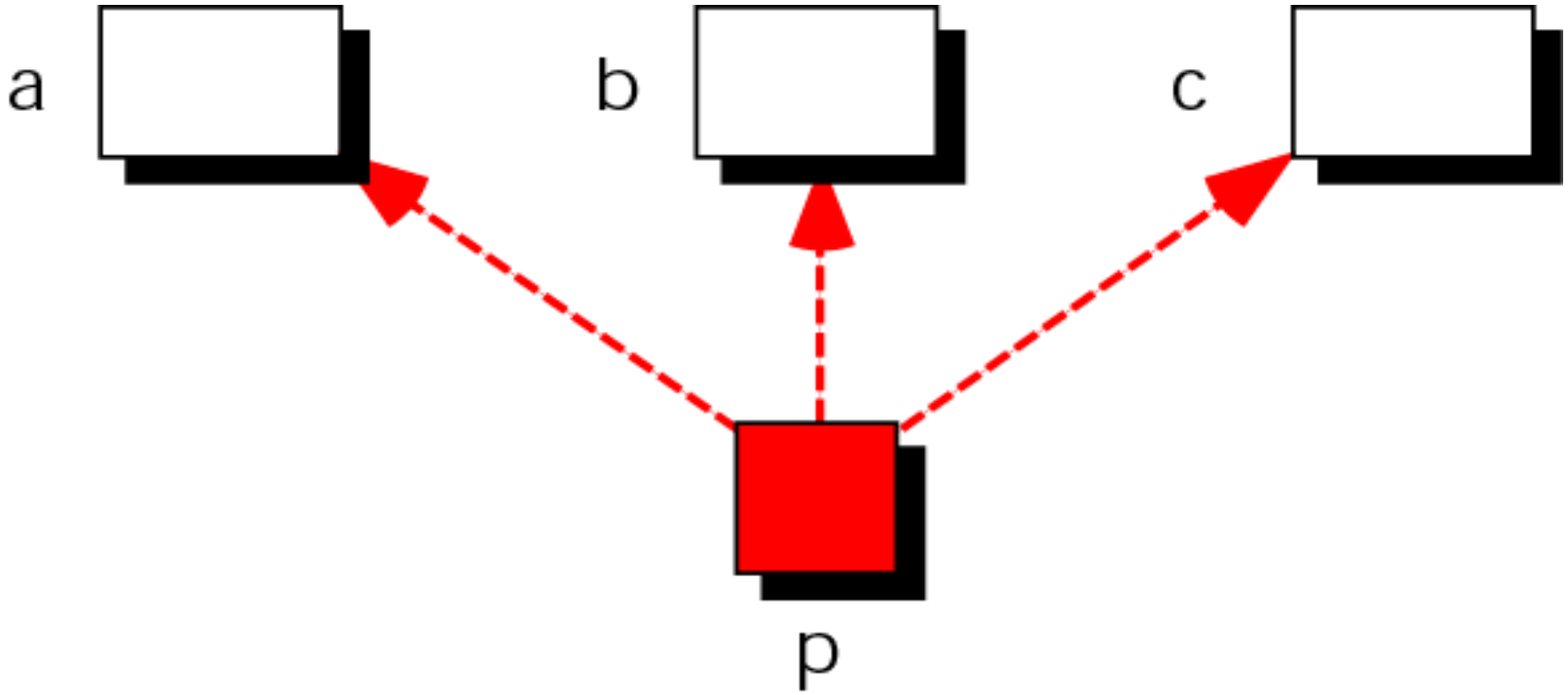


Multiple Pointers to a Variable

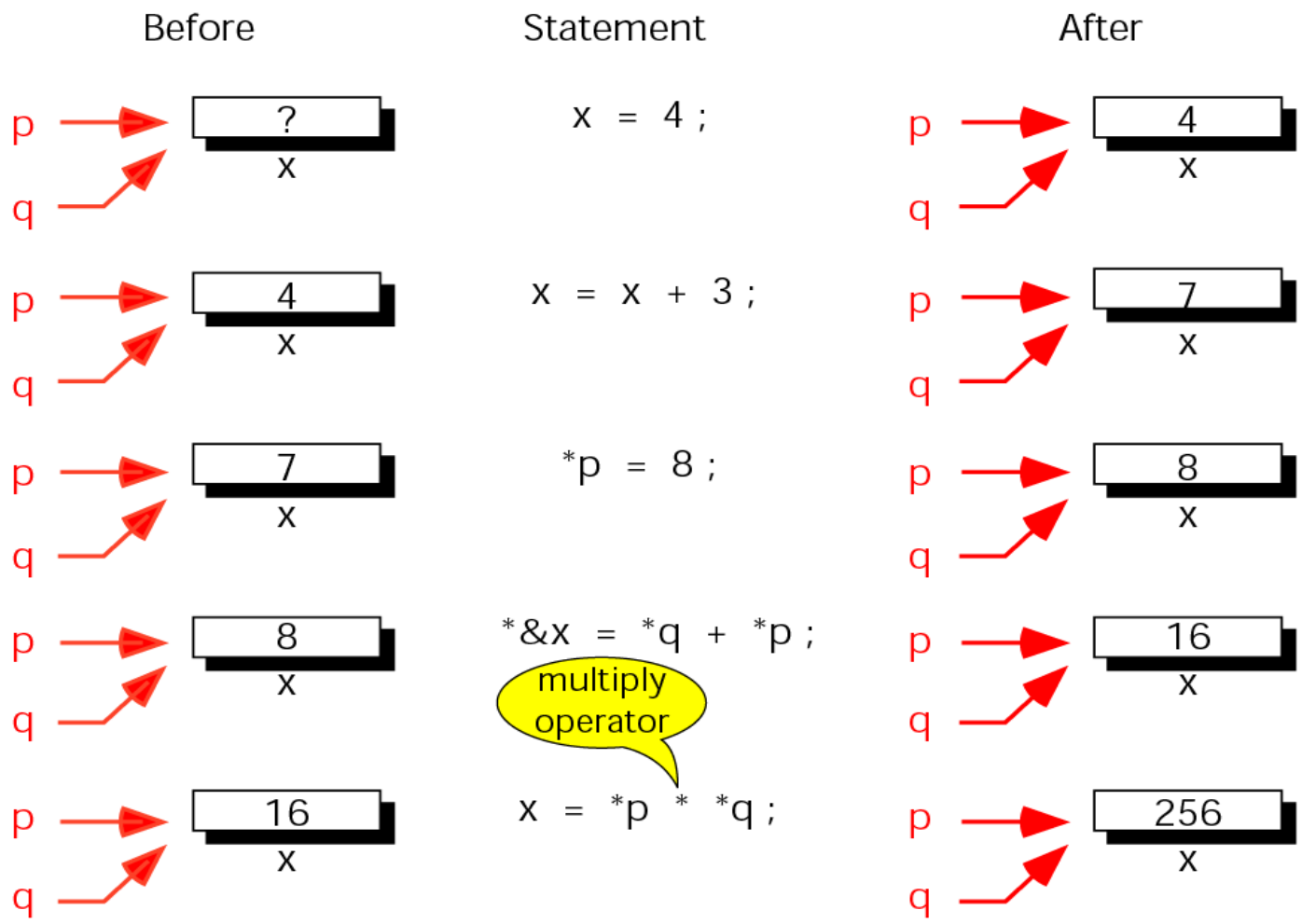




One Pointer With Many Variables



Accessing Variables Through Pointers



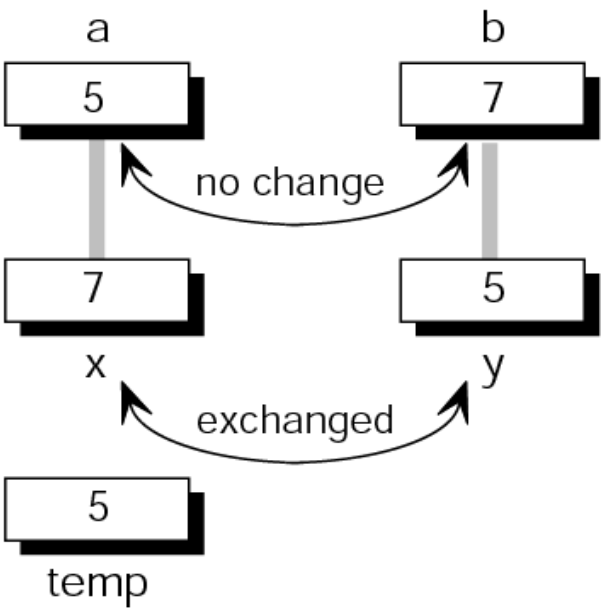
Exchanging Values: by Value

```
int a = 5;
int b = 7;

// Pass by value
exchange (a, b);
```

```
void exchange (int x, int y)
{
    int temp = x;
    x        = y;
    y        = temp;
    return;
} // exchange
```

(a) Original values unchanged



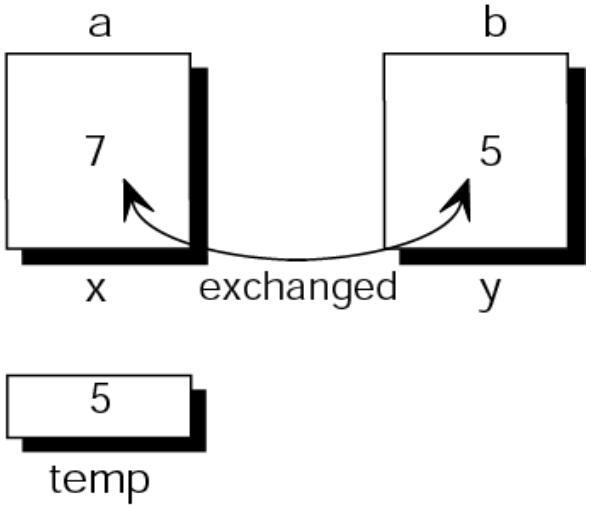
Exchanging Values by Reference

```
int a = 5;
int b = 7;

// Pass by reference
exchange (a, b);
```

```
void exchange (int& x, int& y)
{
    int temp = x;
    x        = y;
    y        = temp;
    return;
} // exchange
```

(b) Original values exchanged



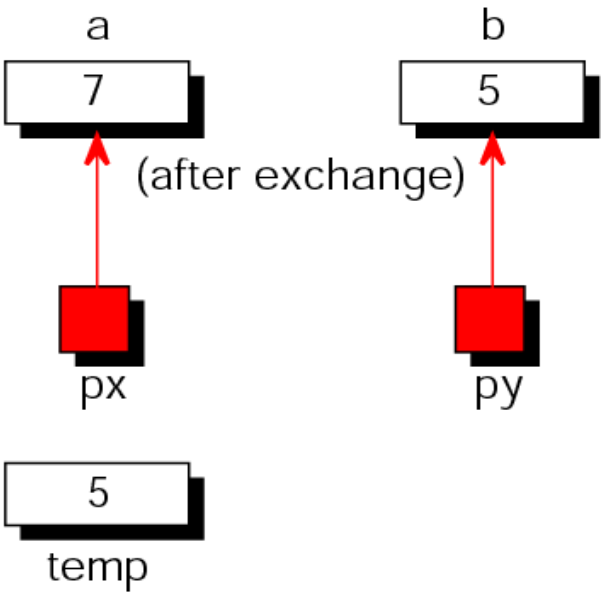
Exchanging Values with Pointers

```
int a = 5;
int b = 7;

// Passing pointers
exchange (&a, &b);
```

```
void exchange (int* px, int* py)
{
    int temp    = *px;
    *px         = *py;
    *py         = temp;
    return;
} // exchange
```

(c) Original values exchanged



Pointers Passed to a Function: Example

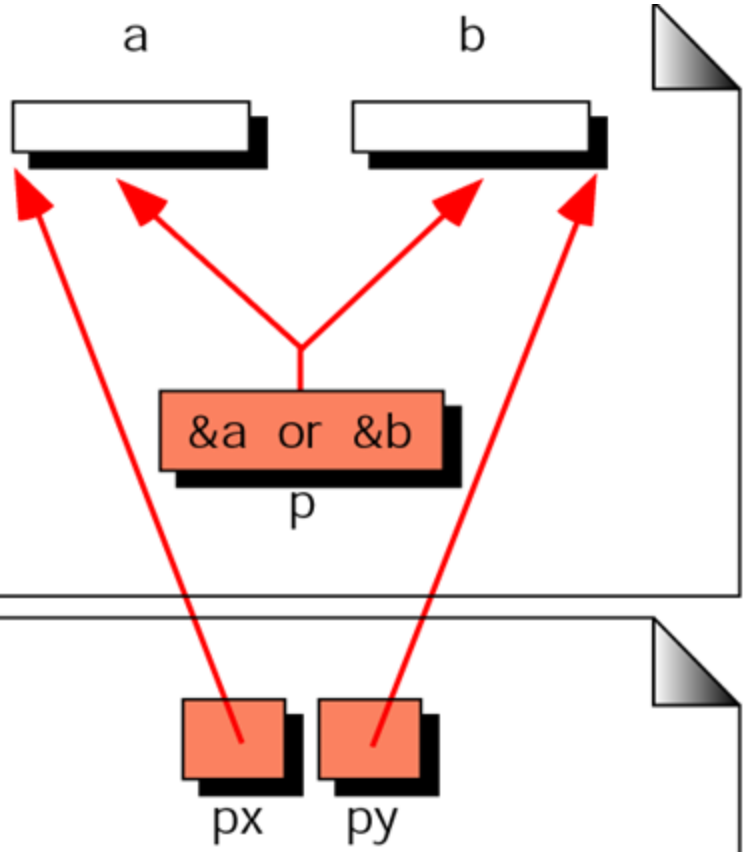
```
#include <stdio.h>
void swapNum(int*, int*);
int main(void) {
        int a = 10;
        int b = 20;
        swapNum(&a, &b);
        printf("A is %d and B is %d\n", a, b);
        system("pause");
        return 0;
}
void swapNum(int *i, int *j) {
        int temp = *i;
        *i = *j;
        *j = temp;
}
```

Output: A is 20 and B is 10

Functions Returning Pointers

```
int*  smaller ( int*, int* ) ;
int  main  ( )
{
   int   a ;
   int   b ;
   int  *p ;
   ...

   p = smaller ( &a, &b );
   ...
}  // main
```



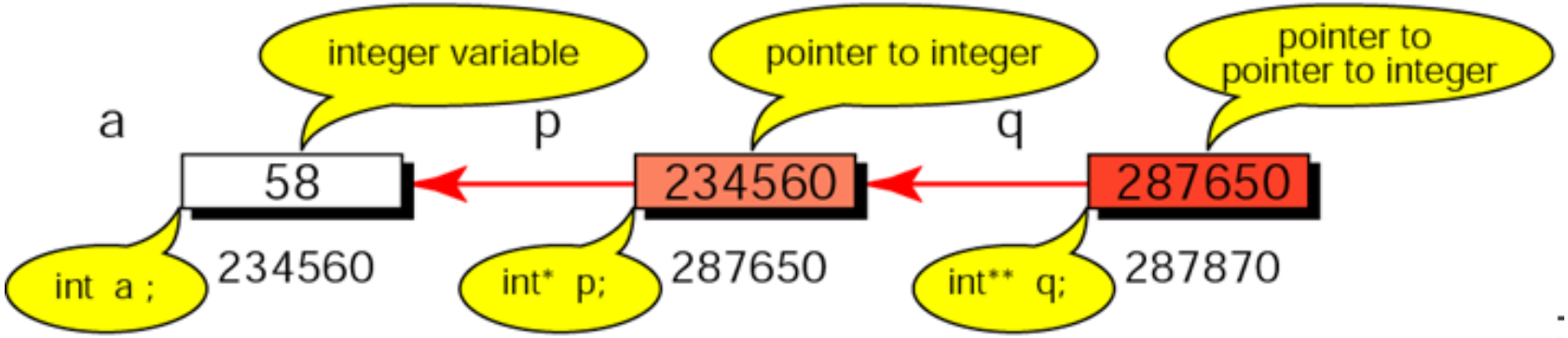

```
int*   smaller  (int*  px, int*  py )
{
   return  ( *px < *py ? px : py ) ;
}  // smaller
```

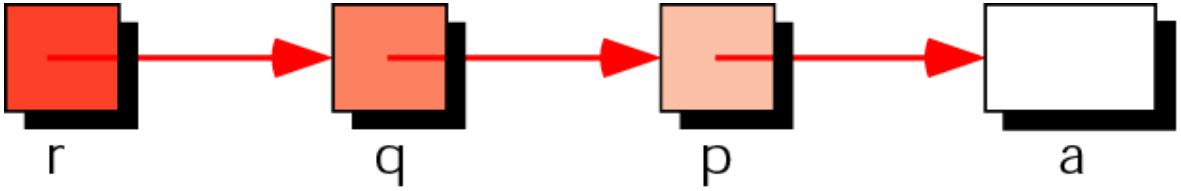
Note:

It is a serious error to return a pointer to a local variable.

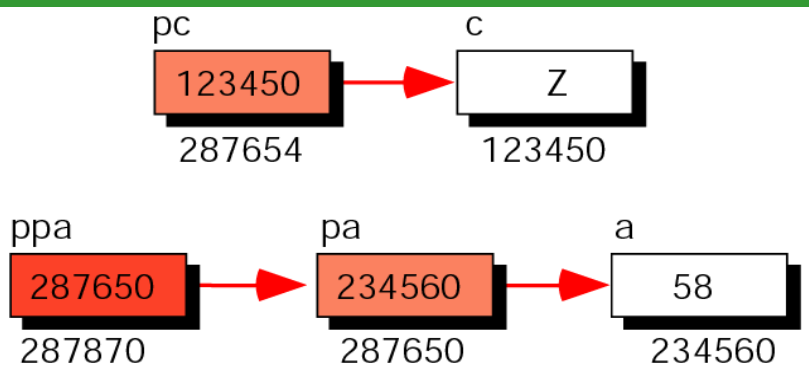
Pointers to Pointers

```
// Definitions
int      a ;
int*     p ;
int**    q ;
```





Pointer Compatibility



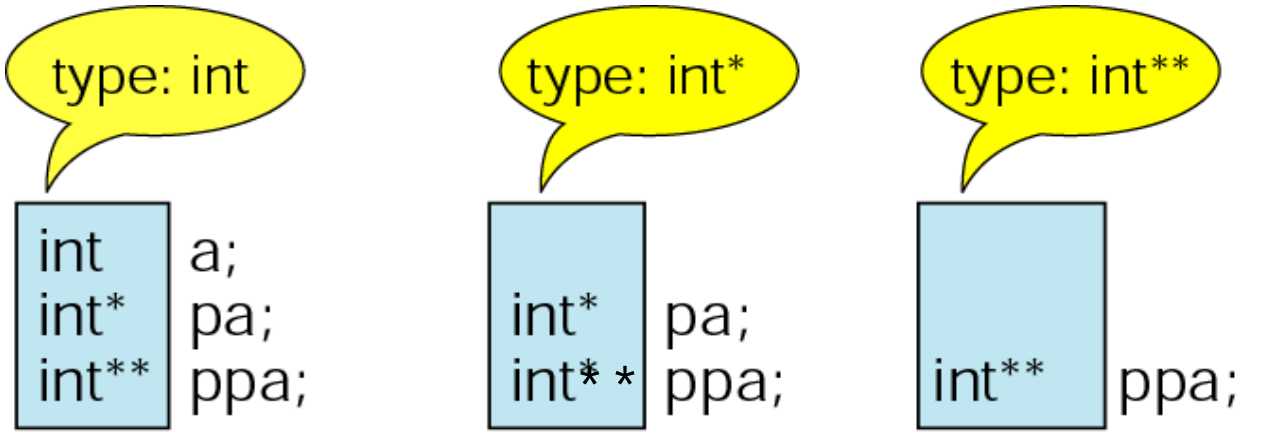

```
char     c = 'z' ;
char*    pc ;

int      a = 58 ;
int*     pa ;
int**    ppa ;

pc     = &c ;          //  Good and valid
pa     = &a ;          //  Good and valid
ppa    = &pa ;         //  Good and valid

//  The following are invalid and will generate errors
pc     = &a ;          //  Different types
ppa    = &a ;          //  Different levels
```

Pointer Types Must Match




```
a     = 4;
*pa   = 4;
**ppa = 4;
```

```
pa   = &a;
*ppa = &a;
```

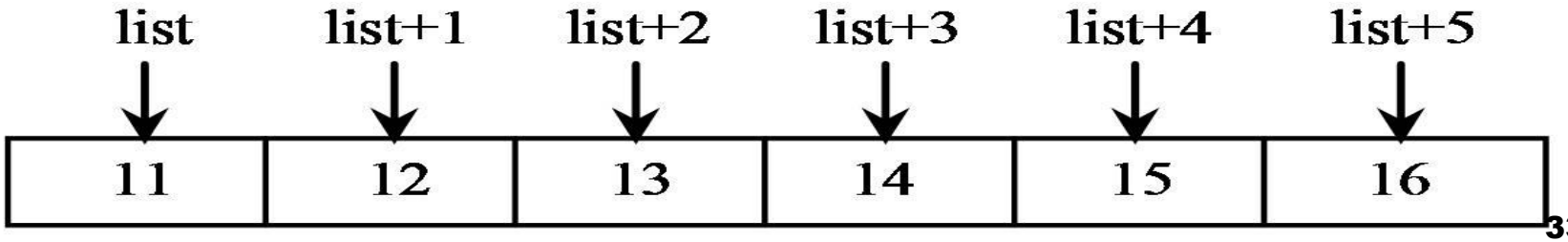
```
ppa = &pa;
```

Arrays and Pointers

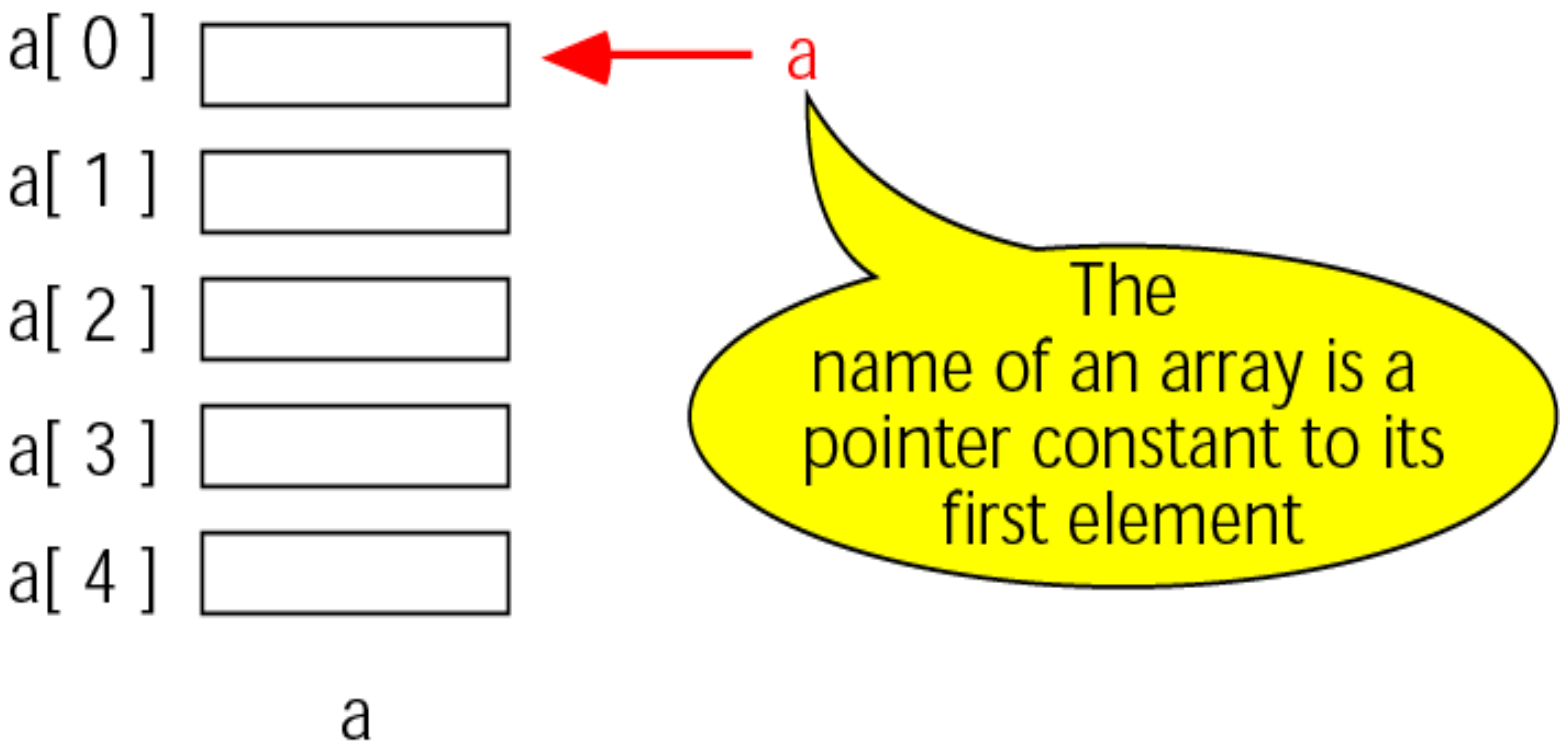
An array variable without a bracket and a subscript actually represents the starting address of the array.

An array variable is essentially a pointer.
Suppose you declare an array of int value as follows:
int list[6] = {11, 12, 13, 14, 15, 16};

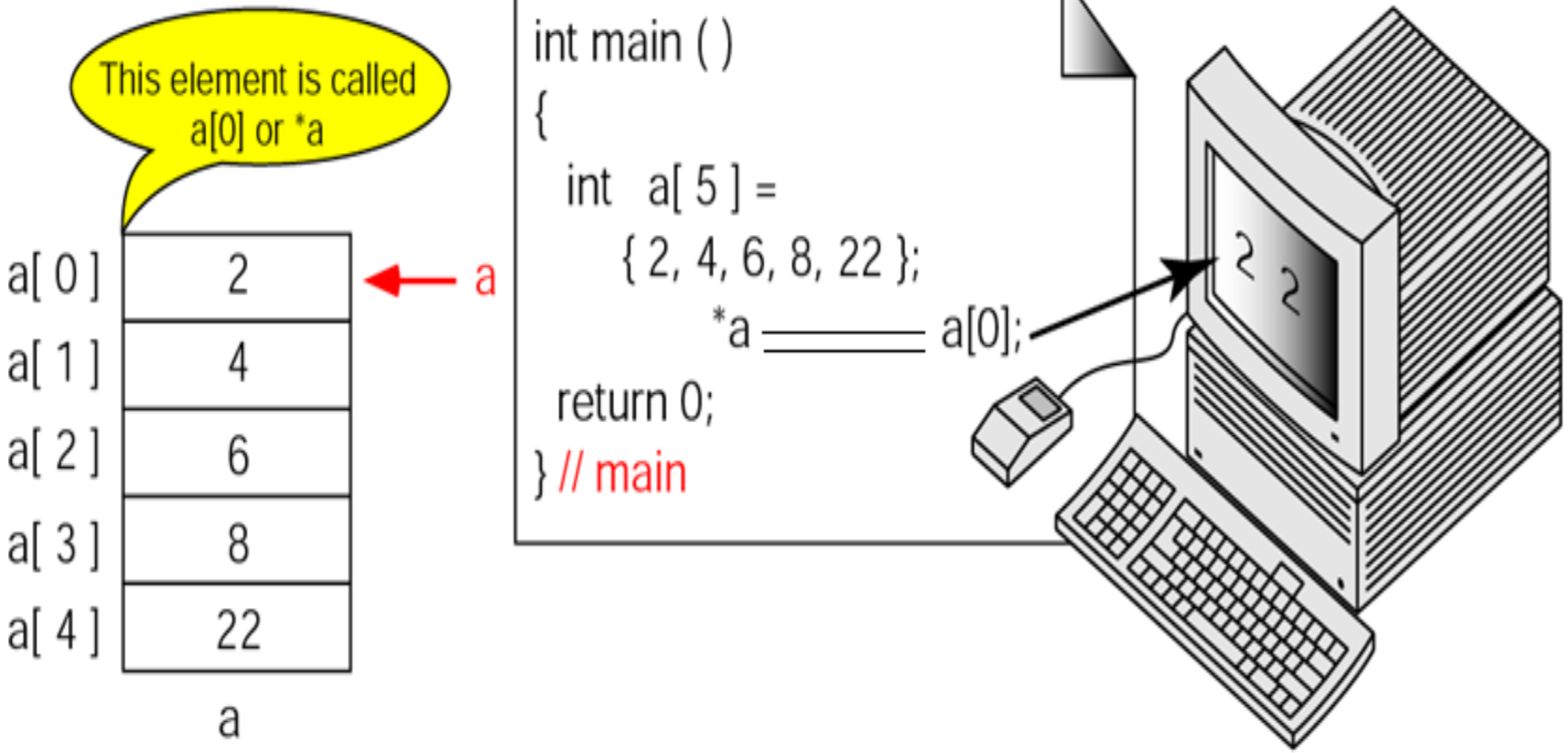
*(list + 1) is different from *list + 1. The dereference operator (*) has precedence over +.
So, *list + 1 adds 1 to the value of the first element in the array, while *(list + 1) dereference the element at address (list + 1) in the array.



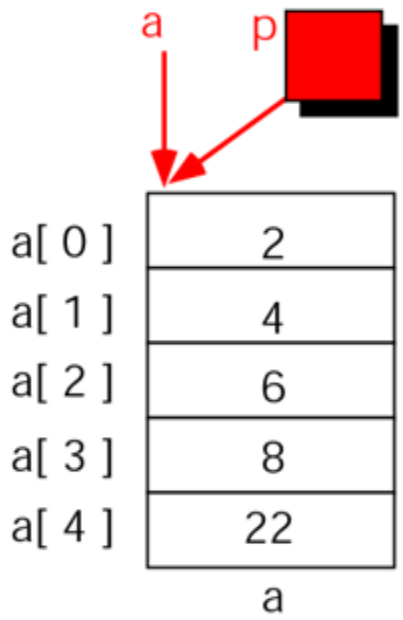
Pointers to Arrays



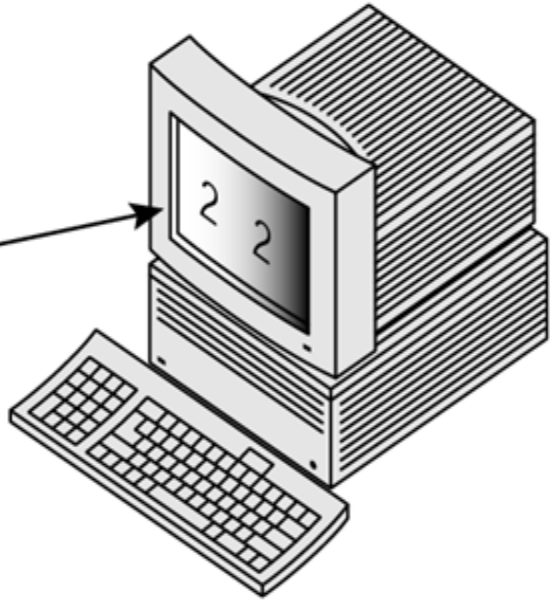
Dereference of Array Name



Array Names as Pointers



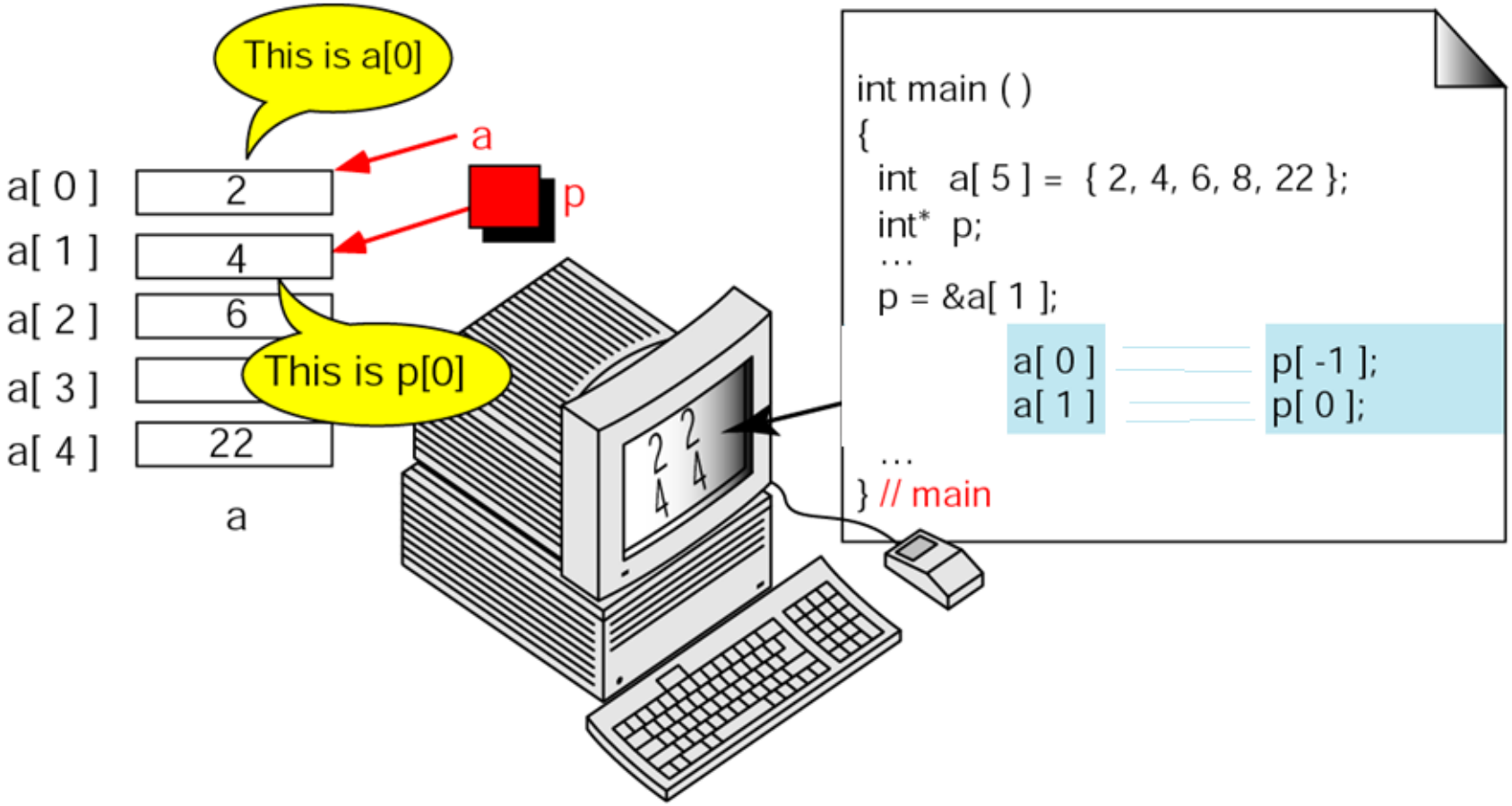

```
int main ( )
{
  int  a[ 5 ]  = { 2, 4, 6, 8, 22 };
  int *p       =  a;
  int  i       =  0;
  ...
                a[ i ] _____ *p;
  ...
  return 0;
} // main
```



Note:

To access an array, a pointer to the first element can be used instead of the name of the array.

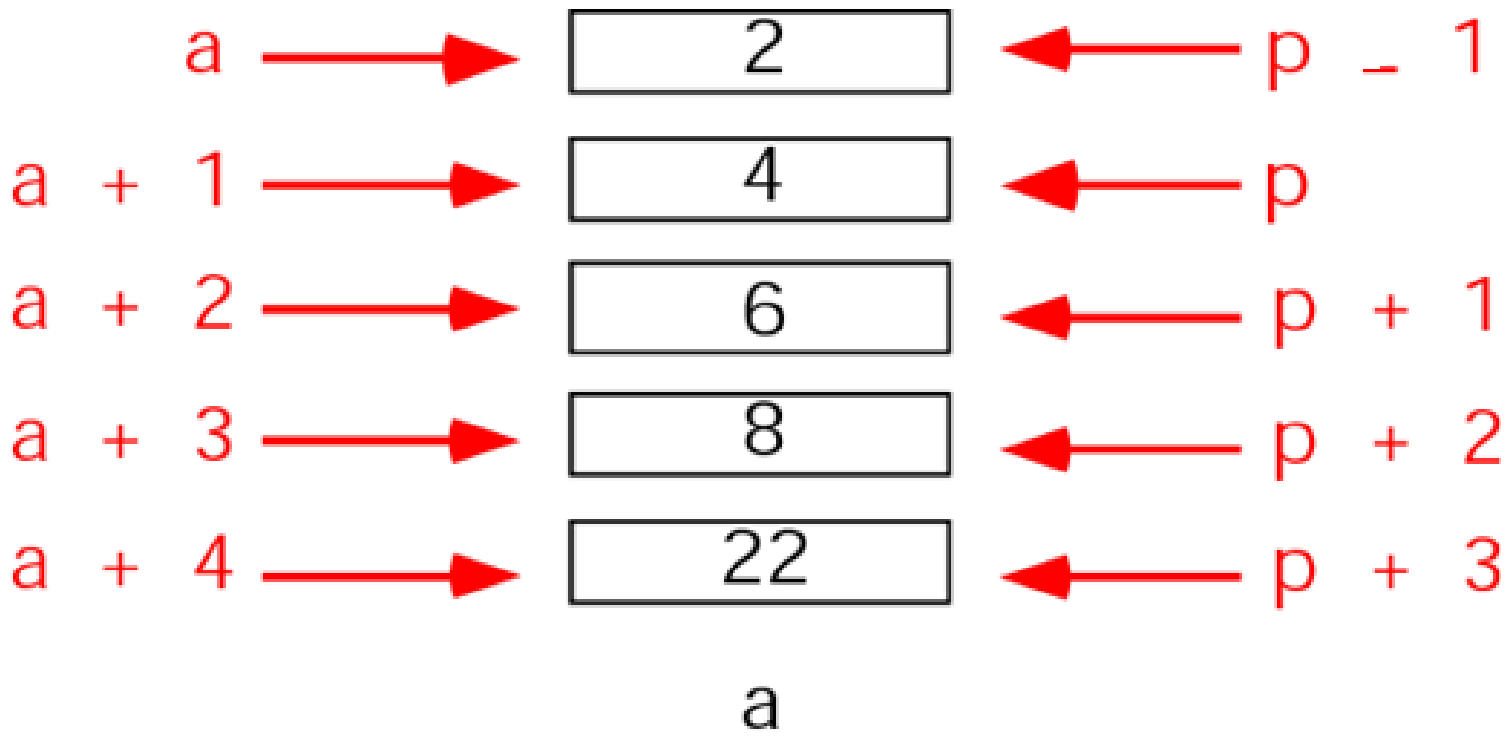
Multiple Array Pointers



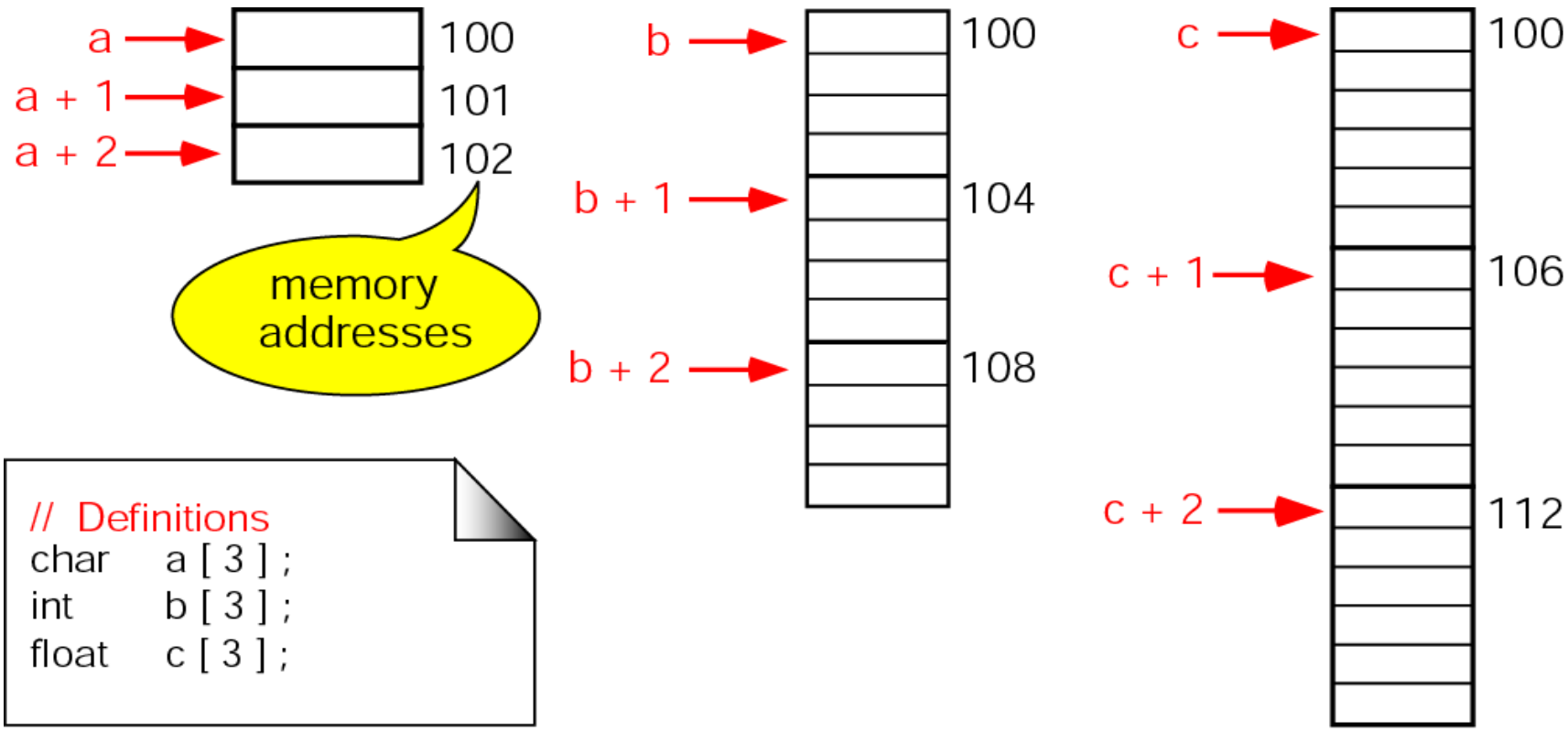
Note:

Given pointer, p, p ± n is a pointer to the value n elements away.

Pointer Arithmetic

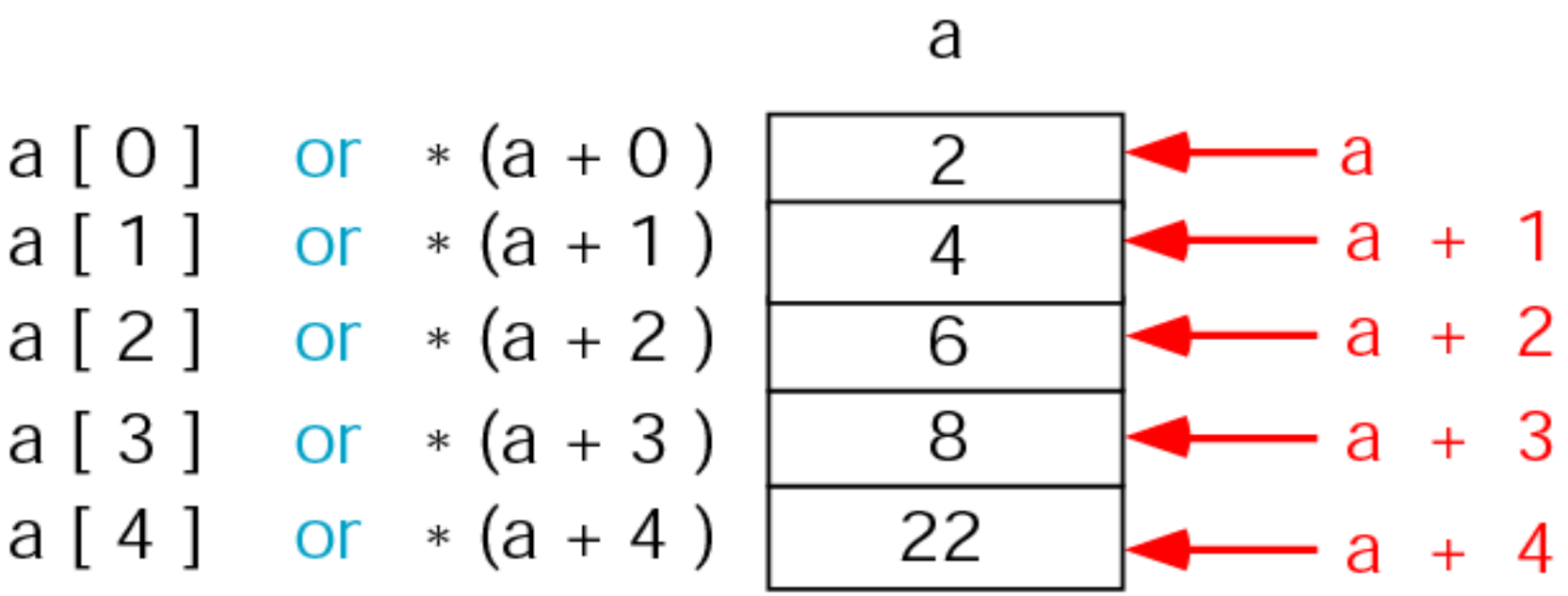


Pointer Arithmetic and Different Types



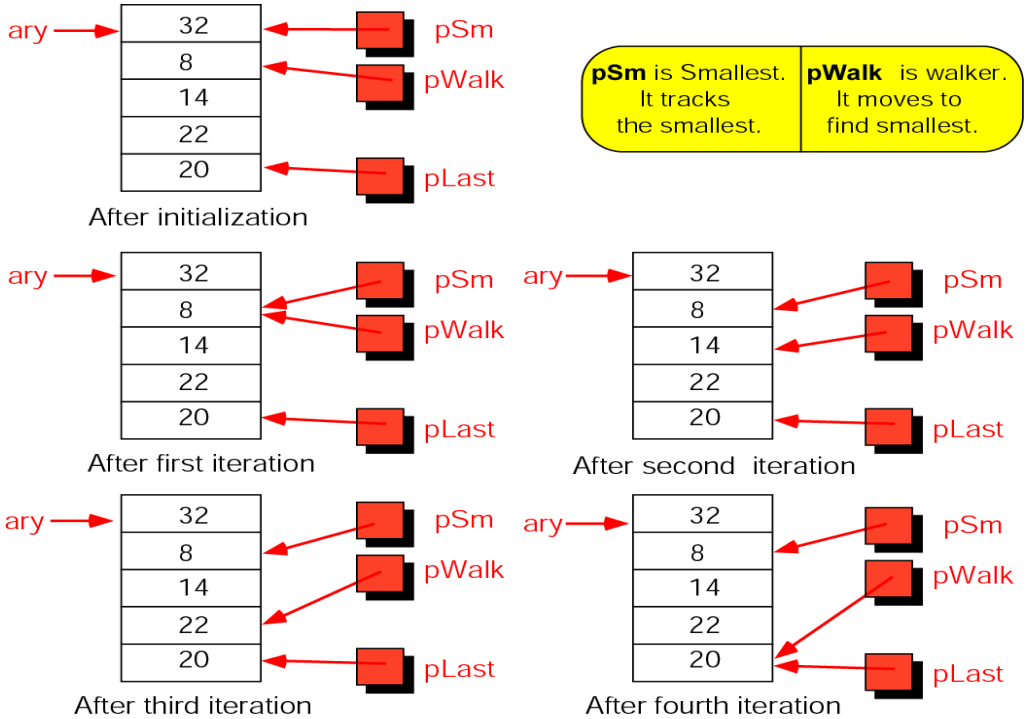

```
// Definitions
char    a [ 3 ] ;
int     b [ 3 ] ;
float   c [ 3 ] ;
```

Dereferencing Array Pointers



* (a + n) is identical to a[n]

Find Smallest



pSm is Smallest. It tracks the smallest.

pWalk is walker. It moves to find smallest.

```
pLast = ary + arySize – 1 ;
for (int  pSm = ary, pWalk = ary + 1 ;
          pWalk <= pLast ;
          pWalk++ )
     if ( *pWalk < *pSm )
               pSm = pWalk ;
```

The End!