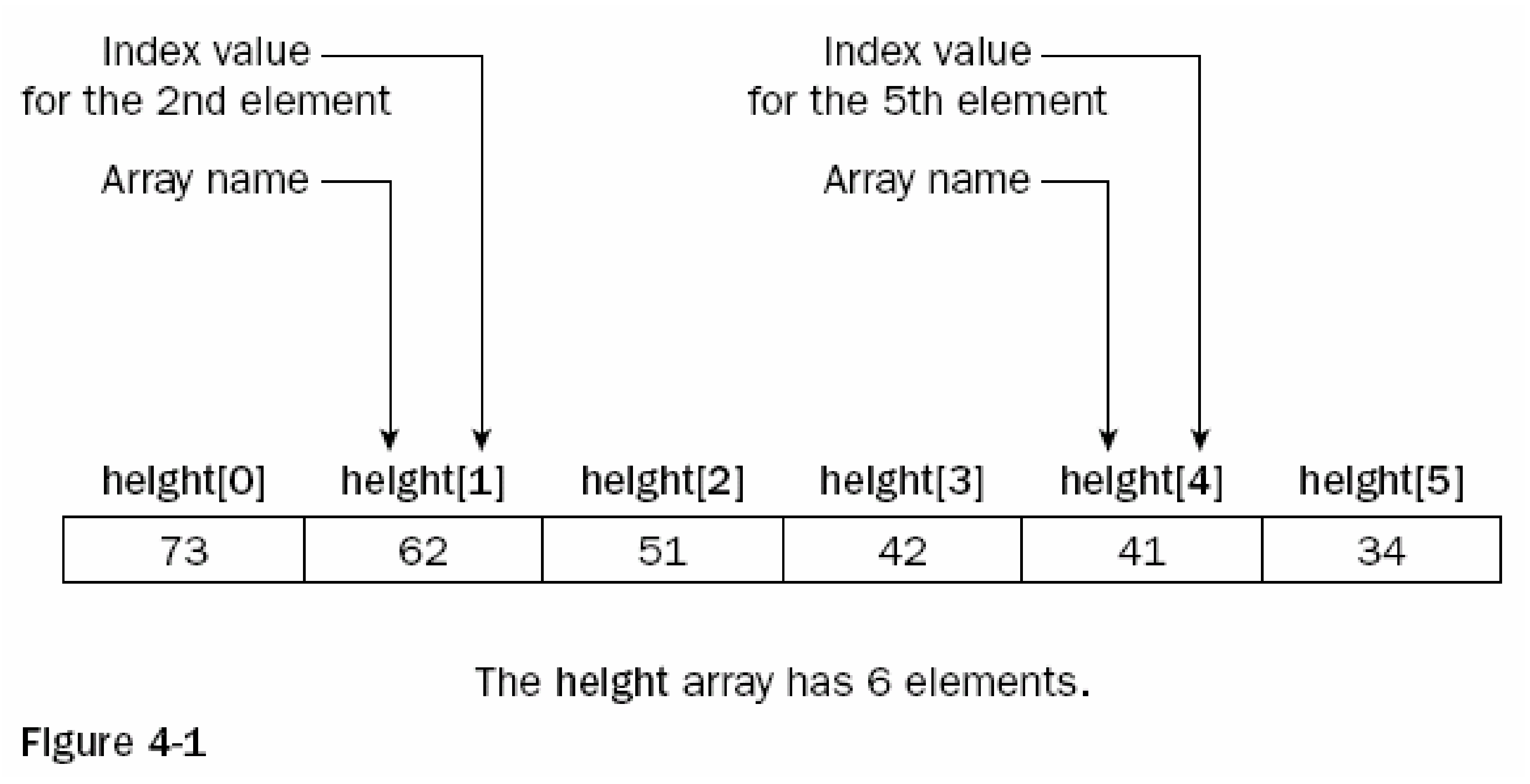
Chapter 4

Arrays, String, and Pointers

Arrays

- To reference several data elements of a particular type with a single variable name.
- Individual items in an array are specified by an index value.
  - The first having the index number 0.
  - The last having the index number N-1.
- All the elements of an array are stored in a contiguous block of memory.

Figure 4-1



Index value for the 2nd element
Array name
Index value for the 5th element
Array name
height[0]
height[1]
height[2]
height[3]
height[4]
height[5]
73
62
51
42
41
34
The **height** array has 6 elements.
Figure 4-1

Declaring Arrays

- `int height[6];`
  - Because each `int` value occupies 4 bytes in memory, the whole array requires 24 bytes.

- `double horsepower[10];`
  - Q: How many bytes will be required for this array?

- `const int MAX = 20;`
- `double miles [ MAX ];`

Using Arrays

- Ex4_01.cpp on P.164

- `cin >> gas[count];`
- `cin >> miles[count];`

- `cout << (miles[i] – miles[i-1])/gas[i];`

- If you use illegal index values, there are no warnings produced either by the compiler or at run-time.
  - `MAX=20`, so index values 0~19 are legal.
  - `gas[-1]` and `gas[30]` are illegal

Initializing Arrays

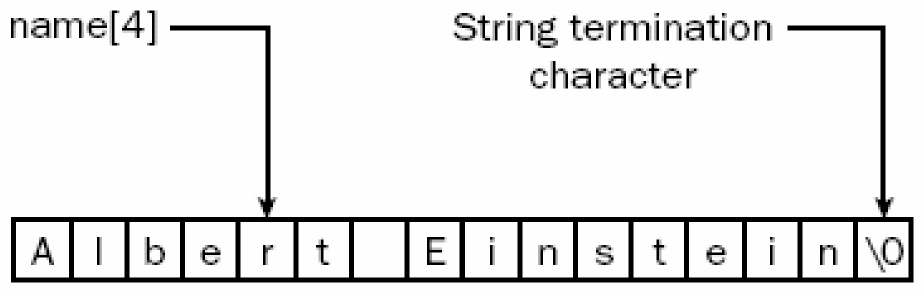
- To initialize an array in its declaration, you put the initializing values separated by commas between braces
  - `int apple = 10;`
  - `int miles[5] = {1019, 1650, 2197, 2749, 3273};`
- The array elements for which you didn’t provide an initial value is initialized with zero.
  - This isn’t the same as supplying no initializing list.
  - Without an initializing list, the array elements contain junk values.

Initializing Arrays (2)

- A convenient way to initialize a whole array to zero is simply to specify a single initializing value as 0.
  - `int data[100] = { 0 };`
- You may also omit the dimension of an array of numeric type:
  - `int value[] = { 2, 3, 4 } ;`
  - The number of elements in the array is determined automatically.

Character Arrays and Strings

- An array of type **char** is called a character array.
  - It is generally used to store a character string.
  - A string terminates with a null character, which is defined by the escape sequence '\0'.
    - It is a byte with all bits as zero.



Each character in a string occupies one byte

char name[] = "Albert Einstein";

Figure 4-2

String Input

- const int MAX = 20;
- char name [MAX];
- cin.getline(name, MAX, ‘\n’);

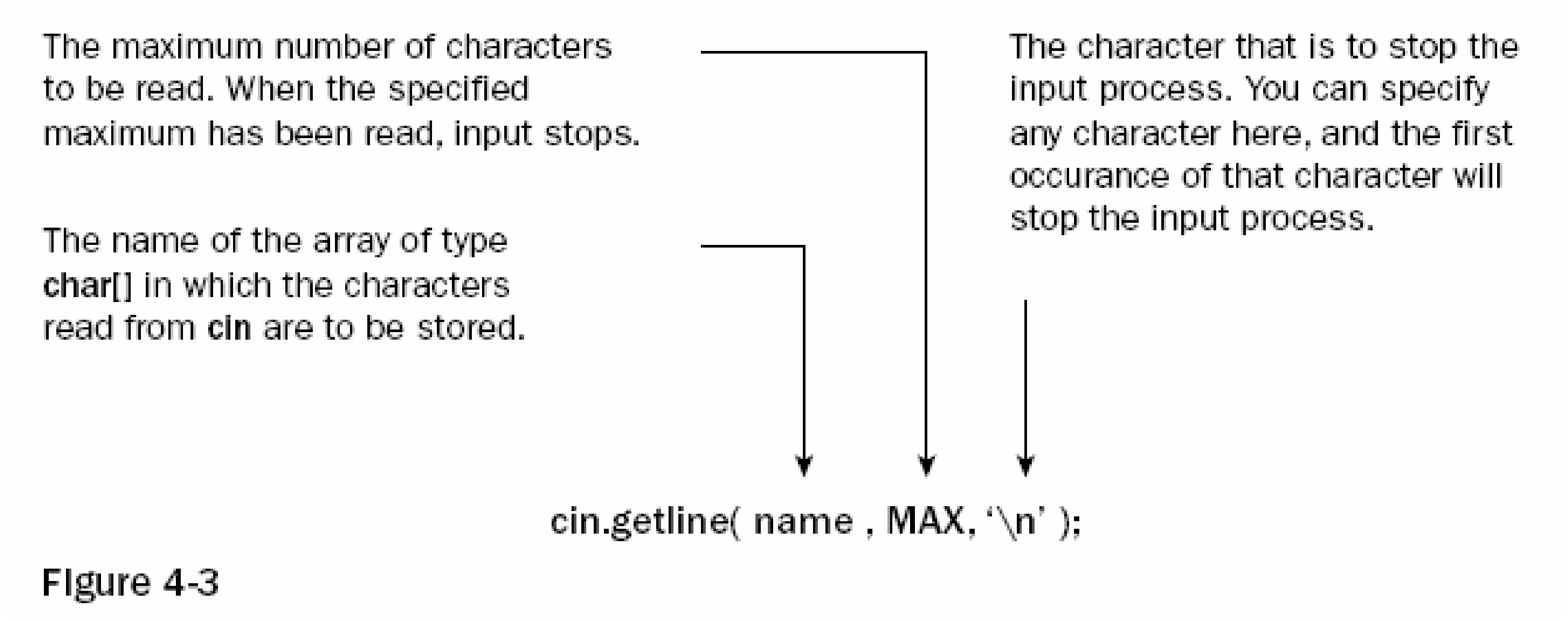

The maximum number of characters to be read. When the specified maximum has been read, input stops.
The name of the array of type **char**[] in which the characters read from **cin** are to be stored.
The character that is to stop the input process. You can specify any character here, and the first occurance of that character will stop the input process.
cin.getline(name , MAX, ‘\n’);

Figure 4-3

String Input (2)

- It is your responsibility to ensure that the array is large enough for any string you might subsequently want to store.
  - Q: Can the array "`char name[20]`" store the string "12345678901234567890"?
- The maximum number of characters read is MAX-1 (rather than MAX),
  - to allow for the `'\0'` character to be appended.
- The `'\n'` isn't stored in the input array name

String Input (3)

- You may also use **cin** to input a string, but please note that the delimiter of **cin** is *whitespaces*.
  - Q: If you supply "*Albert Einstein*", what will "`cin >> name`" store into the string name?

- Ex4_03.cpp on P.170

Multidimensional Array

- An array can also have more than one index value,
  - in which case it is called a **multidimensional array**.
  - `double matrix[3][7];`
  - `matrix[2][4] = 10.7`
- Note that a two-dimensional array in native C++ is essentially a one-dimensional array of one-dimensional array.

Initializing Multidimensional Arrays

- Initialize a two-dimensional array

```
int data [2][4] = {
                      { 1,  2,  3,  5},
                      { 7, 11, 13, 17}
                  };
```

- You can omit initializing values in any row

```
int data [2][4] = {
                      { 1,  2,  3 },
                      { 7, 11}
                  };
```

Initializing Multidimensional Arrays (2)

- Initializing a whole array with zeros.

```
int data[2][4][6] = { 0 };
```

- Storing Multiple Strings (2-dim char array)

```
char stars[][80] = { “Robert Redford”,
                     “Hopalong Cassidy”,
                     “Lassie”,
                     “Slim Pickens”,
                     “Boris Karloff”,
                     “Oliver Hardy”
                   };
```

- Note that you cannot omit both array dimensions. The rightmost dimension(s) must always be defined.

Example: Coin Tossing

- A coin has two sides – Head/Tail
  - 0/1
- Repeat tossing the coin 20 times
- Count the occurrences of Head and Tail, respectively.

Random Number Generator

- `rand()`
  - The rand function returns a pseudorandom integer in the range 0 to RAND_MAX (32767)

```
// Print 5 random numbers.
for (int i = 0; i < 5; i++ )
    cout << rand() << endl;
```

Seed of `rand()`

- With the same seed, the program will get the same result at each execution.
- Use `srand()` and choose the current time as the seed.

```
#include <time.h>
srand((unsigned) time(NULL));
for (int i = 0; i < 5; i++ )
    cout << rand() << endl;
```

Recursive Definition

- Fibonacci sequence
  - F[0] = 0, F[1] = 1, F[n] = F[n-1] + F[n-2]
  - 0 1 1 2 3 5 8 13 21 34 55 89 144 …
- Lucas sequence
  - L[0] = 2, L[1] = 1, L[n] = L[n-1] + L[n-2]
  - 2 1 3 4 7 11 18 29 47 76 123 199 …
- You may write a program to verify
  - L[n] == F[n+2] - F[n-2]

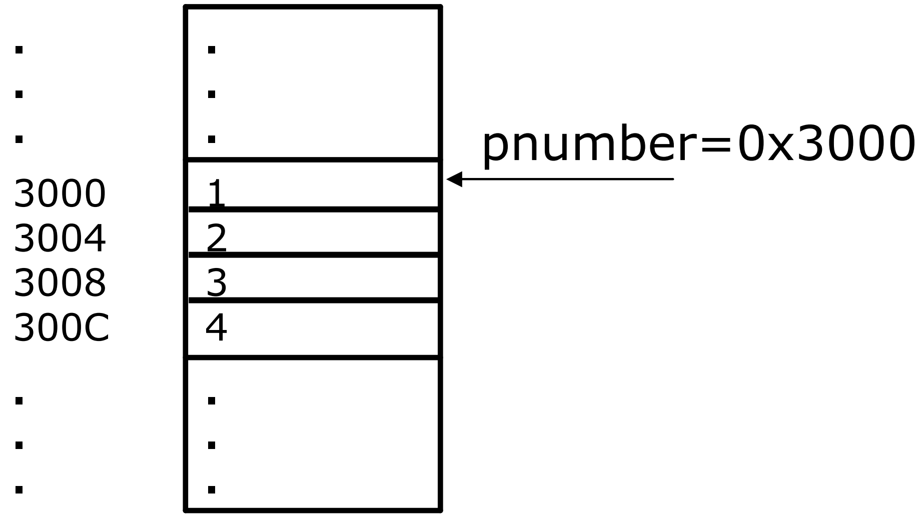
Expected Result

```
Fibonacci sequence:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
Lucas sequence:
2 1 3 4 7 11 18 29 47 76 123 199 322 521 843 1364 2207 3571 5778 9349
=========================
L[ 2]=   3      F[ 4]=   3      F[ 0]=  0       3 == 3 - 0
L[ 3]=   4      F[ 5]=   5      F[ 1]=  1       4 == 5 - 1
L[ 4]=   7      F[ 6]=   8      F[ 2]=  1       7 == 8 - 1
L[ 5]=  11      F[ 7]=  13      F[ 3]=  2       11 == 13 - 2
L[ 6]=  18      F[ 8]=  21      F[ 4]=  3       18 == 21 - 3
L[ 7]=  29      F[ 9]=  34      F[ 5]=  5       29 == 34 - 5
L[ 8]=  47      F[10]=  55      F[ 6]=  8       47 == 55 - 8
L[ 9]=  76      F[11]=  89      F[ 7]= 13       76 == 89 - 13
L[10]= 123      F[12]= 144      F[ 8]= 21       123 == 144 - 21
L[11]= 199      F[13]= 233      F[ 9]= 34       199 == 233 - 34
L[12]= 322      F[14]= 377      F[10]= 55       322 == 377 - 55
L[13]= 521      F[15]= 610      F[11]= 89       521 == 610 - 89
L[14]= 843      F[16]= 987      F[12]=144       843 == 987 - 144
L[15]=1364      F[17]=1597      F[13]=233       1364 == 1597 - 233
L[16]=2207      F[18]=2584      F[14]=377       2207 == 2584 - 377
L[17]=3571      F[19]=4181      F[15]=610       3571 == 4181 - 610
```

```
int n = 0;
const int M = 20;
int L[M] = {2, 1};
int F[M] = {0, 1};

for (n=2; n<M; n++)
{
     L[n] = L[n-1] + L[n-2];
     F[n] = F[n-1] + F[n-2];
}

cout << "Fibonacci sequence: " << endl;
for (n=0; n<M; n++)
     cout << F[n] << " ";
cout << endl;

cout << "Lucas sequence: " << endl;
for (n=0; n<M; n++)
     cout << L[n] << " ";

cout << endl << "=========================" << endl;

 for (n=2; n<M-2; n++)
     cout << "L[" << setw(2) << n << "]=" << setw(4) << L[n] << "\t"
              << "F[" << setw(2) << n+2 << "]=" << setw(4) << F[n+2] << "\t"
              << "F[" << setw(2) << n-2 << "]=" << setw(3) << F[n-2] << "\t"
              << L[n] << (L[n]==F[n+2]-F[n-2]?" == ":" != ")
              << F[n+2] << " - " << F[n-2]
              << endl;
```

Indirect Data Access with Pointers

- Each memory location which you use to store a data value has an address.
- A pointer is a variable that stores an address of another variable (of a particular type).
  - e.g., the variable **`pnumber`** is a pointer
  - It contains the address of a variable of type `int`
  - We say **`pnumber`** is of type 'pointer to `int`'.



Declaring Pointers

- To declare a pointer of type int, you may use either of the following statements:
  - `int* pnumber;`
  - `int *pnumber;`
- You can mix declarations of ordinary variables and pointers in the same statement:
  - int* pnumber, number = 99;
    - Note that number is of type `int` instead of `pointer to int`.
- It is a common convention in C++ to use variable names beginning with p to denote pointers.

Initializing Pointers

- int number = 0;
- int* pnumber = &number;

- int* pnumber = NULL;
- int* pnumber = 0;
  - No object can be allocated the address 0, so address 0 indicates that the pointer has no target.
- You could test the pointer
  - if (pnumber == NULL)
      cout << endl << “pnumber is null.”;
  - if (!pnumber)
      cout << endl << “pnumber is null.”;

The Address-Of Operator

- How do you obtain the address of a variable?
  - pnumber = &number;
    - Store address of number in pnumber

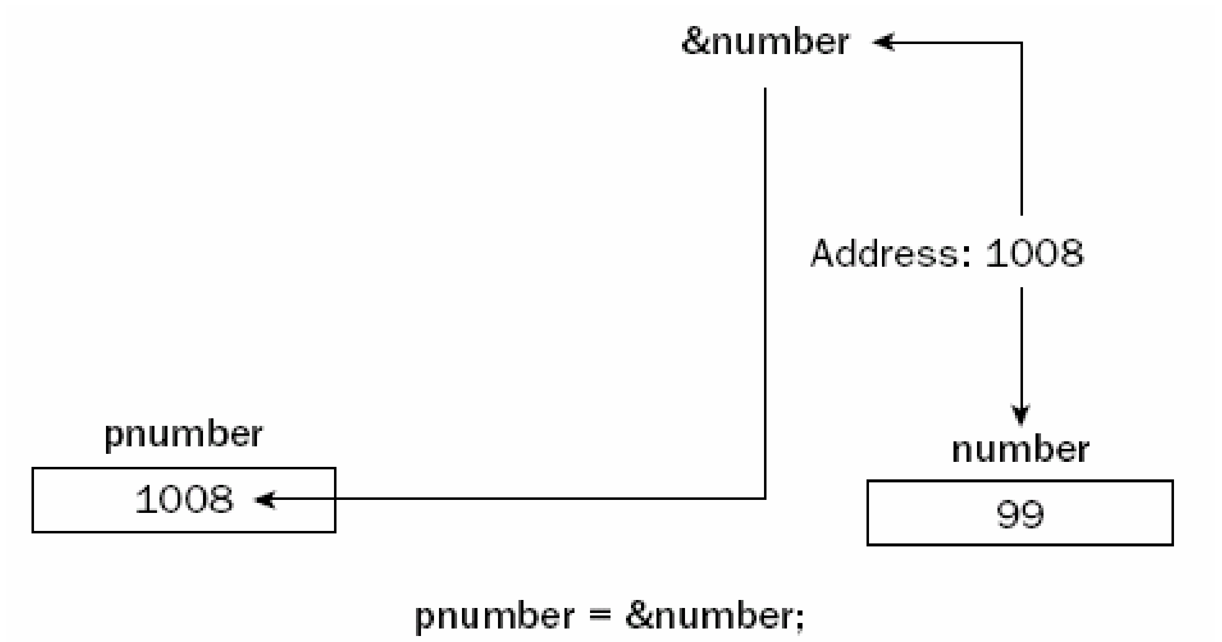


Figure 4-5

The Indirection Operator

- Use the indirection operator *, with a pointer to access the contents of the variable that it points to.
  - Also called the “de-reference operator”
- Ex4_05.cpp on P.177

Why Use Pointers?

- Use pointer notation to operate on data stored in an array
- Allocate space for variables dynamically.
- Enable access within a function to arrays, that are defined outside the function

Pointers to char

  - `char* proverb = “A stitch in time saves nine.”;`
- This looks similar to a char array.
  - `char proverb[] = “A stitch in time saves nine.”;`
- It creates a string literal (an array of type const char)
  - with the character string appearing between the quotes, and terminated with \0
- It also stores the address of the literal in the pointer proverb.

- Compare Ex4_04 on P.173 with Ex4_06 on P.180
  - cout will regard ‘pointer to char’ as a string

Arrays of Pointers

```
char* pstr[] = {   "Rober Redford",
                   "Hopalong Cassidy",
                   "Lassie",
                   "Slim Pickens",
                   "Boris Karloff",
                   "Oliver Hardy"
               };
```

- Using pointers may eliminate the waste of memory that occurred with the array version.
  - In Ex4_04, the char array occupies 80 * 6 = 480 bytes. In Ex4_06, the array occupies 103 bytes.

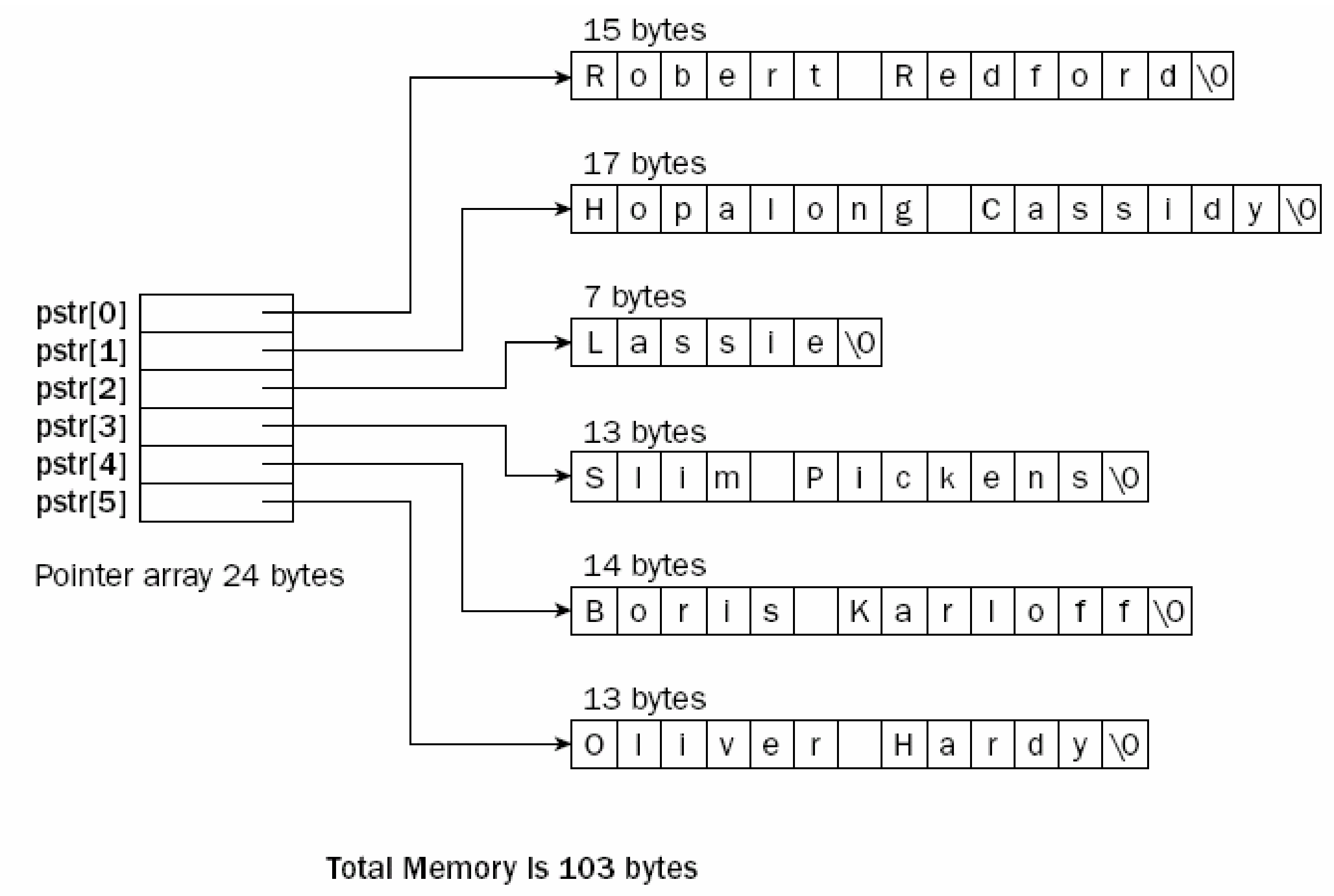


Figure 4-7

The sizeof Operator (1)

- One problem of Ex4_07 is that, the number of strings (6) is “hardwired” in the code.
- If you add a string to the list, you have to modify the code to and change it to be 7.
- Can we make the program automatically adapt to however many strings there are?

The sizeof Operator (2)

- The sizeof operator gives the number of bytes occupied by its operand
  - It produces an integer value of type size_t.
  - size_t is a type defined by the standard library and is usually the same as unsigned int.
- Consider Ex4_07
  - cout << sizeof dice;
    - this statement outputs the value 4, because `int` occupies 4 bytes.
  - cout << sizeof(int);
    - You may also apply the `sizeof` operator to a type name rather than a variable
- Ex4_08.cpp can automatically adapts to an arbitrary number of string values.

Pointers and Arrays

- Array names can behave like pointers.
  - If you use the name of a one-dimensional array by itself, it is automatically converted to a pointer to the first element of the array
- If we have
  - double* pdata;
  - double data[5];
- you can write this assignment
  - pdata = data;
    - Initialize pointer with the array address
  - pdata = &data[1];
    - pdata contains the address of the second element

Pointer Arithmetic

- You can perform addition and subtraction with pointers.
- Suppose pdata = &data[2];
  - The expression pdata+1 would refer to the address of data[3];
  - pdata += 1;
    - Increment pdata to the next element
    - The value of pdata will actually increase by sizeof(double) instead of only 1.
  - pdata++;

De-reference a Pointer with Arithmetic

- Assume `pdata` is pointing to `data[2]`,
  - `*(pdata + 1) = *(pdata + 2);`
  - is equivalent to
  - `data[3] = data[4];`

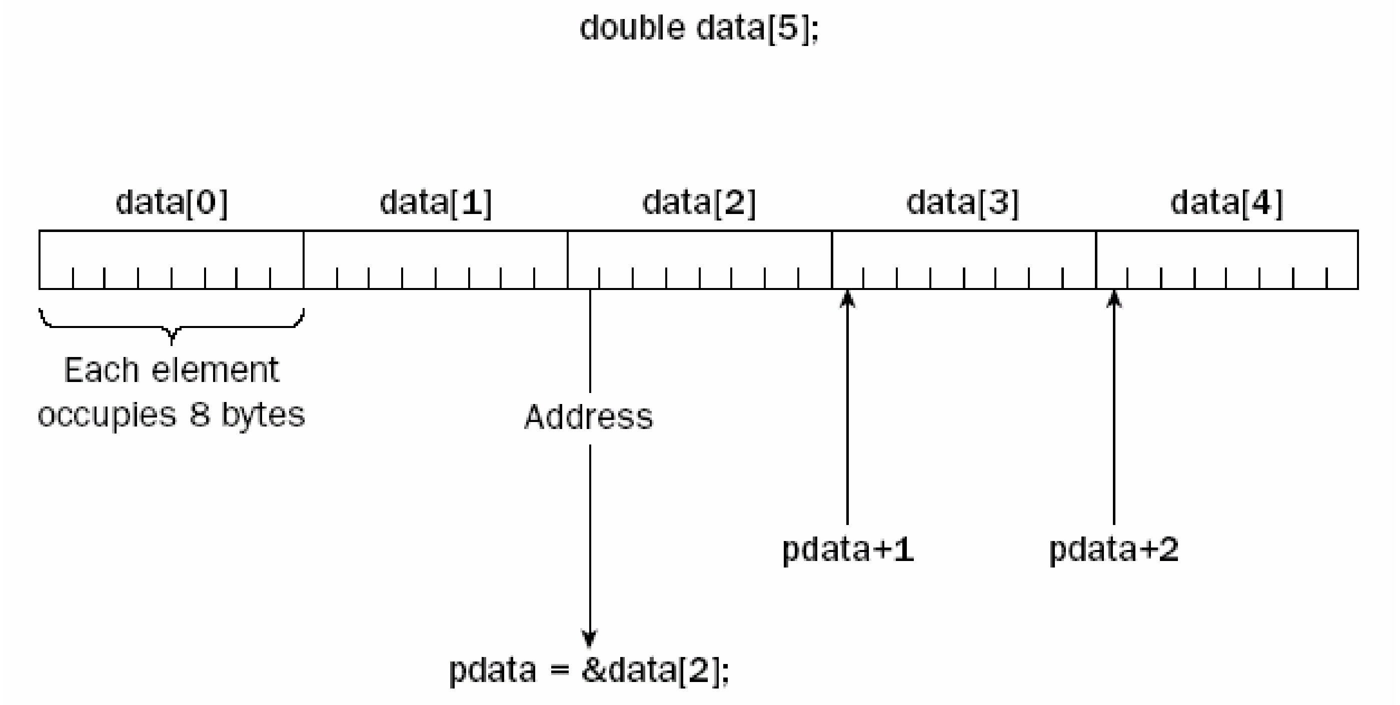


Figure 4-8

Dynamic Memory Allocation

- Sometimes depending on the input data, you may allocate different amount of space for storing different types of variables at execution time

```
int n = 0;
cout << "Input the size of the vector - ";
cin >> n;
int vector[n];
```

error C2057: expected constant expression

Free Store (Heap)

- To hold a string entered by the user, there is no way you can know in advance how large this string could be.

- Free Store - When your program is executed, there is unused memory in your computer.
- You can dynamically allocate space within the free store for a new variable.

The new Operator

- Request memory for a double variable, and return the address of the space
  - `double* pvalue = NULL;`
  - `pvalue = new double;`
- Initialize a variable created by new
  - `pvalue = new double(9999.0);`
- Use this pointer to reference the variable (indirection operator)
  - `*pvalue = 1234.0;`

The delete Operator

- When you no longer need the (dynamically allocated) variable, you can free up the memory space.
  - `delete pvalue;`
    - Release memory pointed to by pvalue
  - `pvalue = 0;`
    - Reset the pointer to 0

- After you release the space, the memory can be used to store a different variable later.

Allocating Memory Dynamically for Arrays

- Allocate a string of twenty characters
  - `char* pstr;`
  - `pstr = new char[20];`
  - `delete [] pstr;`
    - Note the use of square brackets to indicate that you are deleting an array.
  - `pstr = 0;`
    - Set pointer to null

Dynamic Allocation of Multidimensional Arrays

- Allocate memory for a 3x4 array
  - `double (*pbeans)[4];`
  - `pbeans = new double [3][4];`
- Allocate memory for a 5x10x10 array
  - `double (*pBigArray)[10][10];`
  - `pBigArray = new double [5][10][10];`

- You always use only one pair of square brackets following the delete operator, regardless of the dimensionality of the array.
  - `delete [] pBigArray;`