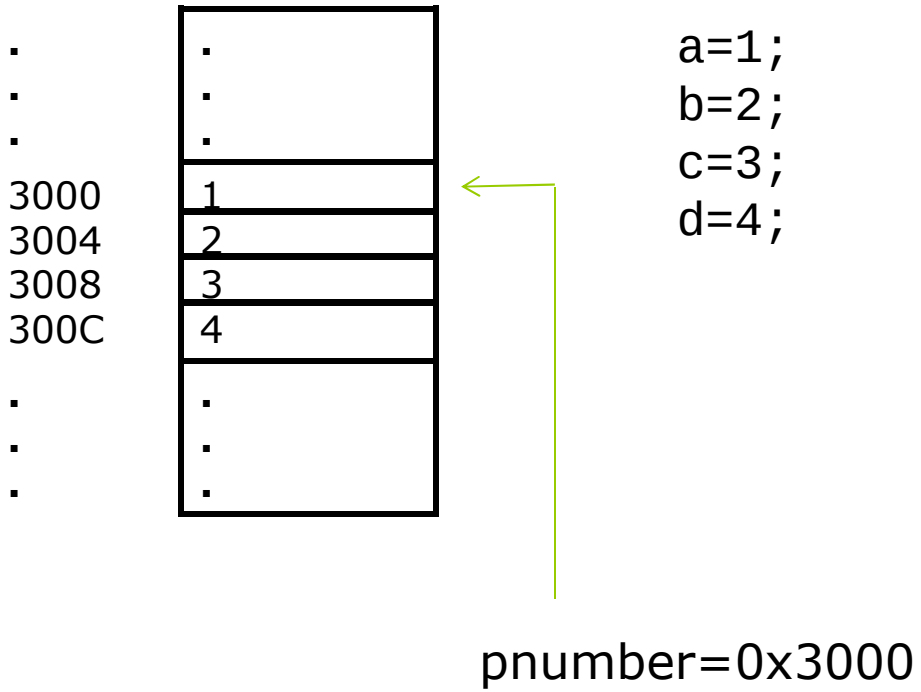
塞下曲

月黑雁飛高，
單于夜遁逃；
欲將輕騎逐，
大雪滿弓刀。

～盧綸

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;`
  - `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.

The Address-Of Operator

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

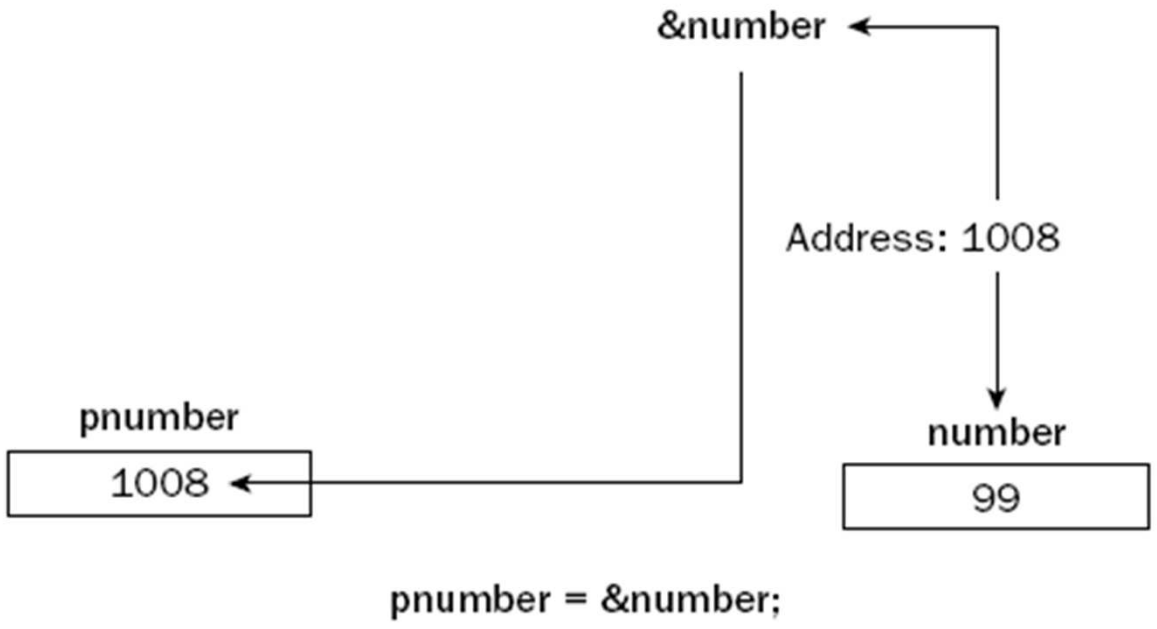


Figure 4-5 (P.182)

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.
  - Visual C++ suggests you to use `nullptr`, but this is not supported by `g++`, so I don't recommend.
- You could test the pointer
  - ```
    if (pnumber == NULL)
      cout << endl << "pnumber is null.";
    ```
  - ```
    if (!pnumber)
      cout << endl << "pnumber is null.";
    ```

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.184
  - *pnumber += 11;
  - number1 += 11;
  - number1 = number1 + 11;

Why Use Pointers? (P.183)

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

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.179 with Ex4_06 on P.186
  - `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.

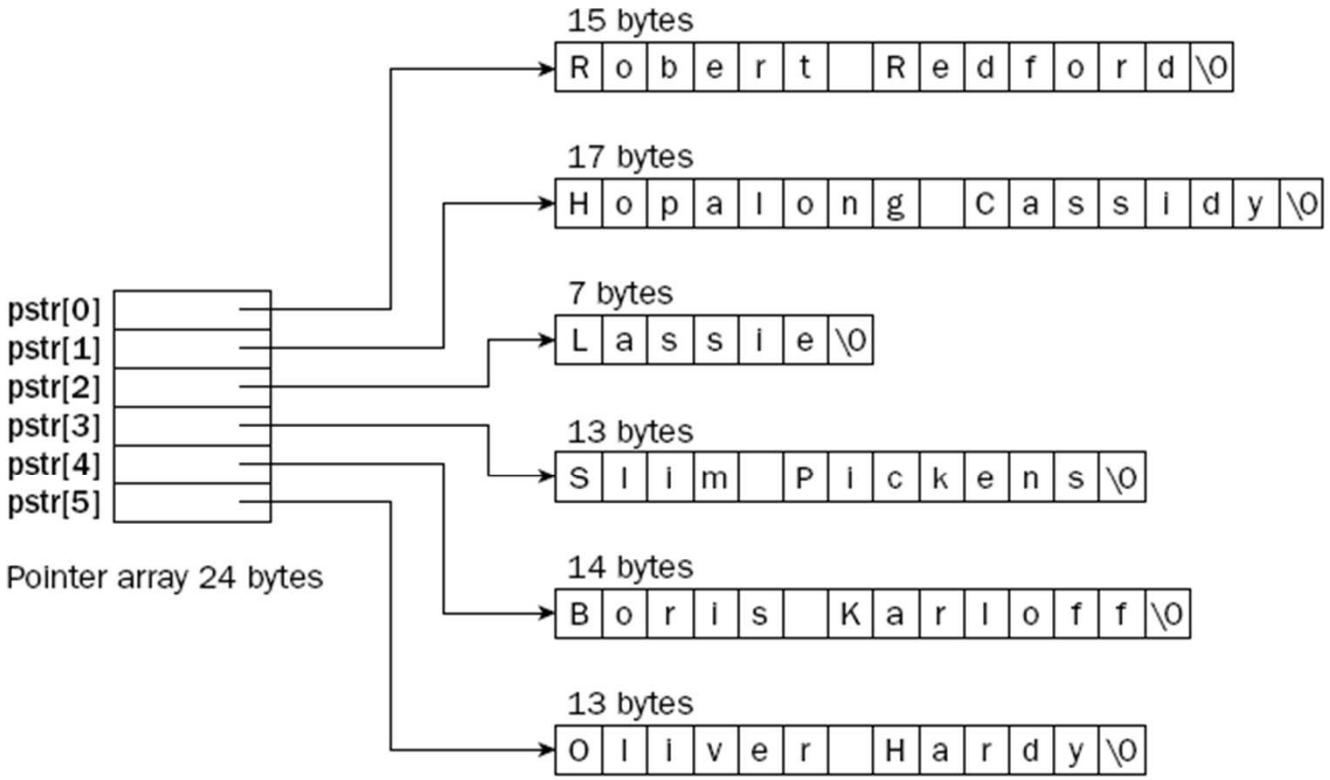


Figure 4-7

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
  - `cout << sizeof pstr;`
    - This statement outpus the value 24, the size of the whole pointer array.
- Ex4_08.cpp can automatically adapt 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];`

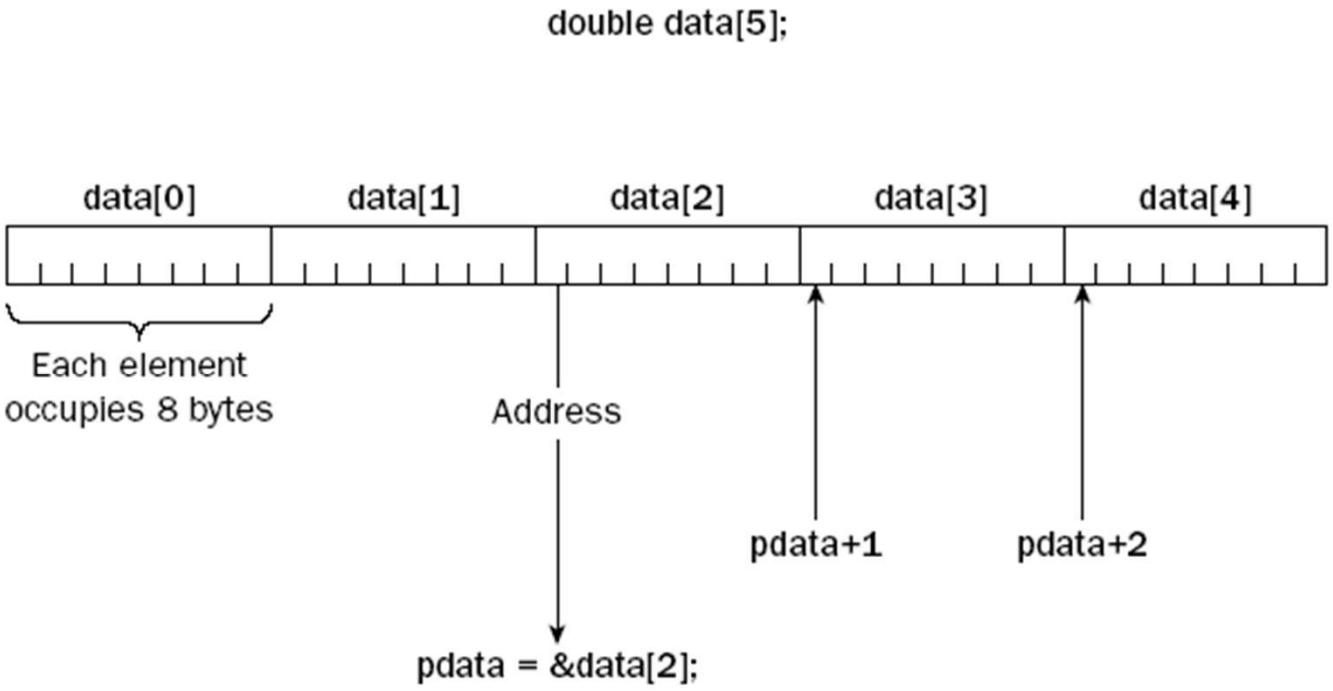


Figure 4-8

(P.195)

Exercises

- Read Ex4_09.cpp and try to draw the flowchart manually. Re-write it by accessing the elements by array indices instead of pointers.
- Modify Ex4_08.cpp to test the sizeof() function. Try to measure the size of a string array, an integer array, and so on.