Resumen

Una forma de resolver este problema sería leer los N números en una matriz, ordenar los matriz en orden decreciente mediante algún algoritmo simple, como la clasificación de burbujas, y luego regresar el elemento en la posición k. Un algoritmo algo mejor podría ser leer los primeros k elementos en una matriz y ordenarlos (en orden decreciente). A continuación, cada elemento restante se lee uno por uno. como nuevo Cuando llega el elemento, se ignora si es más pequeño que el elemento k de la matriz. De lo contrario se coloca en su lugar correcto en la matriz, sacando un elemento de la matriz. Cuando el Cuando finaliza el algoritmo, el elemento en la k-ésima posición se devuelve como respuesta.

Ambos algoritmos son fáciles de codificar y le recomendamos que lo haga. La pregunta natural Las preguntas, entonces, son: ¿Qué algoritmo es mejor? Y, lo que es más importante, ¿alguno de los algoritmos es bueno?

¿suficiente? Una simulación utilizando un archivo aleatorio de 30 millones de elementos y k = 15.000.000 demostrará que ninguno de los algoritmos finaliza en un período de tiempo razonable; cada uno requiere varios días de procesamiento informático para terminar (aunque eventualmente con una respuesta correcta). Un método alternativo, analizado en el capítulo 7, proporciona una solución en aproximadamente un segundo. De este modo, Aunque nuestros algoritmos propuestos funcionan, no pueden considerarse buenos algoritmos.

Un segundo problema es resolver un popular acertijo de palabras. La entrada consta de dos matriz dimensional de letras y una lista de palabras. El objetivo es encontrar las palabras en el rompecabezas.

Estas palabras pueden ser horizontales, verticales o diagonales en cualquier dirección. Como ejemplo, el El rompecabezas que se muestra en la Figura 1.1 contiene las palabras esto, dos, gordo y aquello. La palabra esto comienza en la fila 1, columna 1 o (1,1) y se extiende hasta (1,4); dos van de (1,1) a (3,1); la grasa va de (4,1) a (2,3); y eso va de (4,4) a (1,1). Nuevamente, existen al menos dos algoritmos sencillos que resuelven el problema. Para cada

palabra en la lista de palabras, verificamos cada triplete ordenado (fila, columna, orientación) para la presencencia de la palabra. Esto equivale a muchos bucles for anidados, pero es básicamente sencillo.

Alternativamente, para cada cuádruple ordenado (fila, columna, orientación, número de caracteres) que no se salga del final del rompecabezas, podemos comprobar si la palabra indicada está en el lista de palabras. Nuevamente, esto equivale a muchos bucles for anidados. Es posible ahorrar algo de tiempo. si se conoce el número máximo de caracteres en cualquier palabra.

Matematicas

Esta sección enumera algunas de las fórmulas básicas que necesita memorizar o poder derivar. y revisa técnicas básicas de prueba.

XAXB = XA+B XA XB = XA−B (XA) B = XAB XN + XN = 2XN = X2N 2N + 2N = 2N+1

En informática, todos los logaritmos están en base 2 a menos que se especifique lo contrario.

XA = B si y solo si logX B = A

De esta definición se derivan varias igualdades convenientes.

logA B = logC B logC A; A, B, C > 0, A = 1

Ejemplo Let X = logC B, Y = logC A, and Z = logA B. Entonces, por la definición de logaritmos, CX = B, CY = A, and AZ = B. Combinando estas tres igualdades se obtiene

B = CX = (CY)

Z. por ende, X = YZ, which implies Z = X/Y, proving the theorem.

Los objetos se declaran como tipos primitivos En C++ clásico, un objeto se declara como un tipo primitivo. Por lo tanto, lo siguiente es legal declaraciones de un objeto IntCell: IntCell obj1; // constructor de parámetros cero IntCell obj2(12); // constructor de un parámetro Por otra parte, lo siguiente es incorrecto: IntCell obj3 = 37; // El constructor es explícito IntCell obj4( ); // declaración de función La declaración de obj3 es ilegal porque el constructor de un parámetro es explícito. Él

sería legal de otra manera. (En otras palabras, en C++ clásico una declaración que utiliza el El constructor de parámetros debe utilizar paréntesis para indicar el valor inicial). La declaración

para obj4 indica que es una función (definida en otro lugar) que no toma parámetros y devuelve una IntCell. La confusión de obj4 es una de las razones de la sintaxis de inicialización uniforme que utiliza llaves. Era feo que la inicialización con parámetro cero en una lista de inicialización del constructor (Fig. 1.6, línea 8) requeriría paréntesis sin parámetro, pero la misma sintaxis sería ilegal en otro lugar (para obj4). En C++11, podemos escribir: IntCell obj1; // Constructor de parámetros cero, igual que antes IntCell obj2{ 12 }; // Constructor de un parámetro, igual que antes IntCell obj4{ }; // constructor de parámetros cero La declaración de obj4 es mejor porque la inicialización con un constructor de parámetro cero es ya no es un caso de sintaxis especial; el estilo de inicialización es uniforme.

```
#include <iostream>
 #include <vector>
 using namespace std;
 int main( )
 {
   vector<int> squares( 100 );
   for( int i = 0; i < squares.size( ); ++i )

       squares[ i ] = i * i;

   for( int i = 0; i < squares.size( ); ++i )
     cout << i << " " << squares[ i ] << endl;
 return 0;
 }
```

Sintaxis de clase básica

Una clase en C++ consta de sus miembros. Estos miembros pueden ser datos o funciones. Las funciones se llaman funciones miembro. Cada instancia de una clase es un objeto. Cada El objeto contiene los componentes de datos especificados en la clase (a menos que los componentes de datos sean estático, un detalle que se puede ignorar con seguridad por ahora). Una función miembro se utiliza para actuar sobre un objeto. A menudo, las funciones miembro se denominan métodos. Como ejemplo, la Figura 1.5 es la clase

IntCell. En la clase IntCell, cada instancia de IntCell (un objeto IntCell) contiene un único miembro de datos denominado valoralmacenado. Todo lo demás en esta clase particular es un método. En nuestro ejemplo, hay cuatro métodos. Dos de estos métodos son lectura y escritura. Los otros dos son métodos especiales conocidos como constructores. Describamos algunas características clave.

```
class IntCell
{
public:

IntCell( )
{ storedValue = 0; }

IntCell( int initialValue )
{ storedValue = initialValue; }

int read( )
{ return storedValue; }


void write( int x )
{ storedValue = x; }

private:
int storedValue;
};
```

Aunque la clase funciona tal como está escrita, hay alguna sintaxis adicional que mejora el código. En la Figura 1.6 se muestran cuatro cambios (omitimos los comentarios por motivos de brevedad). Las diferencias son como sigue:

```
1
2 * A class for simulating an integer memory cell.
3
4 class IntCell
5 {
6 public:
7 explicit IntCell( int initialValue = 0 )
8 : storedValue{ initialValue } { }
9 int read( ) const
10 { return storedValue; }
11 void write( int x )
12 { storedValue = x; }
13
14 private:
15 int storedValue;
16 };
```

Los objetos se declaran como tipos primitivos En C++ clásico, un objeto se declara como un tipo primitivo. Por lo tanto, lo siguiente es legal declaraciones de un objeto IntCell: IntCell obj1; // constructor de parámetros cero IntCell obj2(12); // constructor de un parámetro Por otra parte, lo siguiente es incorrecto: IntCell obj3 = 37; // El constructor es explícito IntCell obj4( ); // declaración de función La declaración de obj3 es ilegal porque el constructor de un parámetro es explícito. Él

sería legal de otra manera. (En otras palabras, en C++ clásico una declaración que utiliza el El constructor de parámetros debe utilizar paréntesis para indicar el valor inicial). La declaración

para obj4 indica que es una función (definida en otro lugar) que no toma parámetros y devuelve una IntCell. La confusión de obj4 es una de las razones de la sintaxis de inicialización uniforme que utiliza llaves. Era feo que la inicialización con parámetro cero en una lista de inicialización del constructor (Fig. 1.6, línea 8) requeriría paréntesis sin parámetro, pero la misma sintaxis sería ilegal en otro lugar (para obj4). En C++11, podemos escribir: IntCell obj1; // Constructor de parámetros cero, igual que antes IntCell obj2{ 12 }; // Constructor de un parámetro, igual que antes IntCell obj4{ }; // constructor de parámetros cero La declaración de obj4 es mejor porque la inicialización con un constructor de parámetro cero es ya no es un caso de sintaxis especial; el estilo de inicialización es uniforme.

```
#Include <iostream>
#Include <vector>

using namespace std;

int main()
{
    vector<int> squares(100);

    for(int i = 0; i <square.size(); ++i)
       squares[i] = i * i;

    for( int i = 0; i < squares.size( ); ++i )
      cout << i << " " << squares[ i ] << endl;

    return 0;
}
```

Vecotres y Strings

Las clases de vector y cadena en STL tratan las matrices y cadenas como objetos de primera clase. Un vector sabe qué tan grande es. Dos objetos de cadena se pueden comparar con ==, <, etc. en. Tanto el vector como la cadena se pueden copiar con =. Si es posible, debe evitar el uso de matriz y cadena C++ integradas. Discutimos la matriz incorporada en el Capítulo 3 en el contexto de mostrando cómo se puede implementar el vector.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
```

```cpp
int main() {
    std::vector<std::string> names;


    std::string name;
    while (true) {
        std::cout << "Entra un nombre(O presiona enter para terminar): ";
        std::getline(std::cin, name);

        if (name.empty()) {
            break;
        }

        names.push_back(name);
    }


    if (names.empty()) {
        std::cout << "Ningun nombre puesto, saliendo." << std::endl;
        return 0;
    }


    std::sort(names.begin(), names.end());


    std::cout << "Nombres ordenados:" << std::endl;
    for (const std::string& sortedName : names) {
        std::cout << sortedName << std::endl;
    }

    return 0;
}
```