

2023

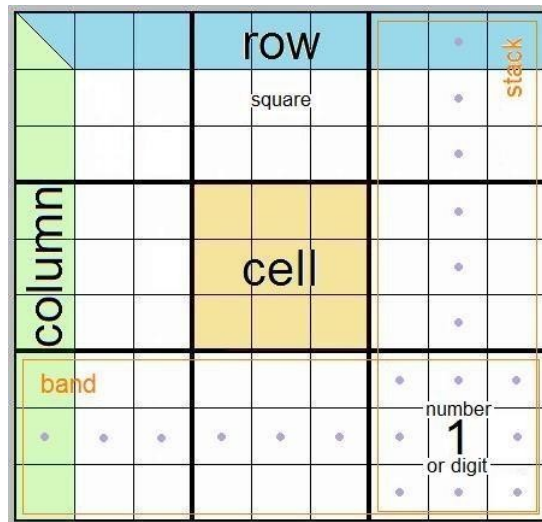
Program Design II

Kun-Ta Chuang
Department of Computer Science and Information Engineering
National Cheng Kung University



Sudoku Validator

- A Sudoku validator reads a Sudoku answer from a file, and then checks if the answer is valid or not.
- A Sudoku answer is a 9×9 grid filled with digits so that **each column**, **each row**, and **each of the nine 3×3 sub-grids (called cells)** that compose the grid contains **all of the digits from 1 to 9**.



8	4	3	5	6	7	2	9	1
5	6	7	1	9	2	4	8	3
2	9	1	4	8	3	7	6	5
1	3	2	9	7	8	6	5	4
9	7	6	3	4	5	8	1	2
4	5	8	6	2	1	3	7	9
7	8	5	2	3	9	1	4	6
3	1	4	7	5	6	9	2	8
6	2	9	8	1	4	5	3	7

Sudoku.h

```
1 #include <iostream>
2 class Sudoku {
3 public:
4     Sudoku();
5     Sudoku(const int init_map[]);
6     void setMap(const int set_map[]);
7     int getElement(int index);
8     bool isCorrect();
9     static const int sudokuSize = 81;
10
11 private:
12     bool checkUnity(int arr[]);
13     int map[sudokuSize];
14 };
```

Sudoku.cpp

```
1 #include "Sudoku.h"
2 using namespace std;
3
4 Sudoku::Sudoku()
5 {
6     for(int i=0; i<sudokuSize; ++i)
7         map[i] = 0;
8 }
9 Sudoku::Sudoku(const int init_map[])
10 {
11     for(int i=0; i<sudokuSize; ++i)
12         map[i] = init_map[i];
13 }
14
15 void Sudoku::setMap(const int set_map[])
16 {
17     for(int i=0; i<sudokuSize; ++i)
18         map[i] = set_map[i];
19 }
```

```
20 int Sudoku::getElement(int index)
21 {
22     return map[index];
23 }
24
25 bool Sudoku::checkUnity(int arr[])
26 {
27     int arr_unity[9]; // counters
28
29     for(int i=0; i<9; ++i)
30         arr_unity[i] = 0; // initialize
31     for(int i=0; i<9; ++i)
32         ++arr_unity[arr[i]-1]; // count
33     for(int i=0; i<9; ++i)
34         if(arr_unity[i] != 1) // all element
35             return false;    // must be 1
36     return true;
37 }
38
```

Sudoku.cpp (cont.)

```
39 bool Sudoku::isCorrect()
40 {
41     bool check_result;
42     int check_arr[9];
43     int location;
44     for(int i=0; i<81; i+=9)    // check rows
45     {
46         for(int j=0; j<9; ++j)
47             check_arr[j] = map[i+j];
48         check_result = checkUnity(check_arr);
49         if(check_result == false)
50             return false;
51     }
52     for(int i=0; i<9; ++i)    // check columns
53     {
54         for(int j=0; j<9; ++j)
55             check_arr[j] = map[i+9*j];
56         check_result = checkUnity(check_arr);
57         if(check_result == false)
58             return false;
59     }
```

```
60     for(int i=0; i<9; ++i)    // check cells
61     {
62         for(int j=0; j<9; ++j)
63         {
64             location = 27*(i/3) + 3*(i%3)
65                       + 9*(j/3) + (j%3);
66             check_arr[j] = map[location];
67         }
68         check_result =
69             checkUnity(check_arr);
70         if(check_result == false)
71             return false;
72     }
```

Sample Input and Sample Output

Number of cases

Map of case #1

Map of case #2

```
>cat su_infile
2
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
8 6 5 3 2 9 4 1 7
2 4 3 1 7 5 8 6 9
1 9 7 6 8 4 5 2 3
3 1 9 2 5 8 6 7 4
4 2 6 7 9 1 3 5 8
5 7 8 4 3 6 1 9 2
7 5 4 9 1 3 2 8 6
6 8 2 5 4 7 9 3 1
9 3 1 8 6 2 7 4 5
```

```
>./sudoku_validate
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
INCORRECT
8 6 5 3 2 9 4 1 7
2 4 3 1 7 5 8 6 9
1 9 7 6 8 4 5 2 3
3 1 9 2 5 8 6 7 4
4 2 6 7 9 1 3 5 8
5 7 8 4 3 6 1 9 2
7 5 4 9 1 3 2 8 6
6 8 2 5 4 7 9 3 1
9 3 1 8 6 2 7 4 5
CORRECT
```

Validation Result

sudoku_validate.cpp

```
1 #include <cstdlib>
2 #include <iostream>
3 #include <fstream>
4 #include "Sudoku.h"
5 #define MAX_CASE 100
6 using namespace std;
7 int main()
8 {
9     int sudoku_in[Sudoku::sudokuSize];
10    Sudoku su[MAX_CASE];
11    ifstream in("su_infile",ios::in);
12    int num_case;
13    in >> num_case;
14    for(int j=0; j<num_case; ++j)
15    {
16        for(int i=0; i<Sudoku::sudokuSize; ++i)
17            in >> sudoku_in[i];    // read in map
18        su[j].setMap(sudoku_in); // set map
19    }
20    for(int j=0; j<num_case; ++j)
21    {    // print out the maps
22        for(int i=0; i<Sudoku::sudokuSize; ++i)
23        {
24            cout << su[j].getElement(i) << " ";
25            if(i % 9 == 8 )
26                cout << endl;
27        }
28        if(su[j].isCorrect()) // validation results
29            cout << "CORRECT\n";
30        else
31            cout << "INCORRECT\n";
32    }
33    return 0;
34 }
```

public static const Data Member

- Note that the size of the array is specified as a **public static const** data member.
 - **public** so that it's accessible to the clients of the class.
 - **const** so that this data member is **constant**.
 - **static** so that the data member is **shared by all objects of the class**
- **static data members** are also known as **class variables**.
- When objects of a class containing **static** data members are created, **all the objects share one copy of the class's static data members**.

Error: Initialization of *const* Data Member

```
> cat -n const1.cpp
```

```
1  class Cls {  
2  public:  
3      Cls(){ x = 3;}  
4      const int x;  
5  };  
6  int main() { return 0; }
```

```
> g++ -o const1 const1.cpp
```

```
const1.cpp: In constructor `Cls::Cls()':
```

```
const1.cpp:3: error: uninitialized member `Cls::x' with `const' type `const int'
```

```
const1.cpp:3: error: assignment of read-only data-member `Cls::x'
```



Initialization of *const* Data Member (cont.)

```
> cat -n const2.cpp
```

```
1  class Cls {  
2    public: const int x = 3;  
3  };  
4  int main() { return 0; }
```

```
> g++ -o const2 const2.cpp
```

```
const2.cpp:2:23: warning: in-class initialization of non-static data member is  
a C++11 extension [-Wc++11-extensions]
```

```
public:  const int x = 3;  
         ^
```

1 warning generated.

```
> cat -n const3.cpp
```

```
1  class Cls {  
2    public: Cls():x(3) {}  
3           const int x;  
4  };  
5  int main() { return 0; }
```

```
> g++ -o const3 const3.cpp
```

```
>
```



Initialization of *static const* Data Member

static_const1.cpp

```
1 class Cls {  
2 public:   Cls():x(3) {}  
3         static const int x;  
4 };  
5 int main() { return 0; }
```



static_const2.cpp

```
1 class Cls {  
2 public:   Cls(){}  
3         static const int x = 3;  
4 };  
5 int main() { return 0; }
```



```
>g++ -o static_const1  
static_const1.cpp  
static_const1.cpp: In constructor  
`Cls::Cls()':  
static_const1.cpp:2: error: `const int  
Cls::x' is a static data member; it can  
only be initialized at its definition
```

```
>g++ -o static_const2  
static_const2.cpp  
>
```

Size of Object with *static const* and *const* Data Members

```
1 #include <iostream>
2 using namespace std;
3 class Cls {
4 public:  Cls():y(4){}
5         static const int x = 3;
6         const int y;
7 };
8 int main()
9 {
10     Cls obj;
11     cout << "sizeof(Cls) = " << sizeof(Cls) << endl;
12     cout << "sizeof(obj) = " << sizeof(obj) << endl;
13     return 0;
14 }
```

Output:
sizeof(Cls) = 4
sizeof(obj) = 4

static Data Member

```
1 #include <iostream>
2 using namespace std;
3
4 class Cls {
5 public:    Cls(){ NumObject++; }
6     static int NumObject;
7 };
8 int Cls::NumObject = 0;
9 int main()
10 {
11     cout << Cls::NumObject << endl;
12     Cls obj1;
13     cout << Cls::NumObject << endl;
14     Cls obj2;
15     cout << obj1.NumObject << endl;
16     cout << obj2.NumObject << endl;
17     return 0;
18 }
```

Just Declaration

Definition (Do not use "static" here.)

Output:

0
1
2
2

static Data Member (cont.)

- A **static** data member **can be accessed within the class** definition and the member-function definitions like any other data member.
- A **public static** data member **can also be accessed outside of the class, even when no objects of the class exist**, using the class name followed by the binary scope resolution operator (::) and the name of the data member.

Replacing Array with *vector*

```
1  #include <vector>
2  #include <cstdlib>
3  #include <iostream>
4  #include <fstream>
5  #include "Sudoku.h"
6  using namespace std;
7  int main()
8  {
9      int sudoku_in[Sudoku::sudokuSize];
10     Sudoku su_tmp;
11     vector<Sudoku> su;
12     ifstream in("su_infile",ios::in);
13     int num_element, num_case;
14     in >> num_case;
15     // num_case is not used in this program
16     cout << "size = " <<
17         su.size() << endl;
18     num_element = 0;
19     while(in >> sudoku_in[num_element++])
20     {
21         // read in map
22         if(num_element >=
23             Sudoku::sudokuSize) {
24             su_tmp.setMap(sudoku_in);
25             num_element = 0;
26             su.push_back(su_tmp);
27         }
28     }
29     cout << "size = " << su.size() << endl;
30     cout << su[0].isCorrect() << endl;
31     for(int i = 1; i<su.size(); ++i)
32         cout << su.at(i).isCorrect() << endl;
33     return 0;
34 }
```

```
> ./sudoku_validate2
size = 0
size = 2
0
1
```

C++ Standard Library

Class Template vector

- C-style pointer-based arrays have great potential for errors and are not flexible

- A program can easily “walk off” either end of an array, because C++ does not check whether subscripts fall outside the range of an array.

```
arr[-1]
```

- Two arrays cannot be meaningfully compared with equality operators or relational operators.

```
if(arr1 == arr2)
```

- When an array is passed to a general-purpose function designed to handle arrays of any size, the size of the array must be passed as an additional argument.

```
func(arr, size)
```

- One array cannot be assigned to another with the assignment operator(s).

```
arr1 = arr2
```


C++ Standard Library

Class Template vector (cont.)

- C++ Standard Library class **template vector** represents a more robust type of array featuring many additional capabilities.
- Standard class template **vector** is **defined in header <vector>** and **belongs to namespace std**.
- **By default, all the elements** of a **vector** object **are set to 0**.
- **vectors** can be defined to store any data type.
- **vector** member function **size** obtain the number of elements in the vector.
- **vector** objects can be compared with one another using the **equality operators**.

```
vector<int> v1;  
vector<Sudoku> v2;
```

```
cout << v.size();
```

```
if(v1 == v2)
```

C++ Standard Library

Class Template vector (cont.)

- You can create a new **vector** object that is initialized with the contents of an existing **vector** by using its **copy constructor**.

```
vector<Sudoku> v2(v1);
```

- You can use the **assignment (=) operator** with **vector** objects.

```
v1 = v2;
```

- You can use square brackets, **[]**, to access the elements in a **vector**. As with C-style pointer-based arrays, **C++ does not perform any bounds checking when vector elements are accessed with square brackets**.

```
v[1];
```

- Standard class template **vector** provides bounds checking in its member function **at**, which “throws an exception” if its argument is an invalid subscript.

```
v.at(1);
```

Sorting a Vector with Insertion Sort

```
1  #include <vector>
2  #include <iomanip>
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      const int size = 8;
9      int init_array[size] =
10         {64, 24, 13, 9, 7, 23, 34, 47};
11      vector<int> v(size);
12      int insert, moveItem;
13      cout << "Unsorted array:\n";
14      for(int i=0; i<size; ++i)
15      {
16          v.at(i) = init_array[i];
17          cout << setw(4)<< v.at(i);
18      }
19      cout << endl;
20
21      cout << "Step-by-step:\n";
22      for(int next=1; next<size; ++next)
23      {
24          insert = v.at(next);
25          moveItem = next;
26          while((moveItem>0) && (v.at(moveItem-1) > insert))
27          {
28              v.at(moveItem) = v.at(moveItem-1);
29              --moveItem;
30          }
31          v.at(moveItem) = insert;
32          for(int i=0; i<size; ++i)
33              cout << setw(4) << v.at(i);
34          cout << endl;
35      }
36
37      return 0;
38 }
```

Sorting a Vector with Insertion Sort (cont.)

```
22 for(int next=1;next<size;++next)
23 {
24     insert = v.at(next);
25     moveItem = next;
26     while((moveItem>0) &&
27           (v.at(moveItem-1) > insert))
28     {
29         v.at(moveItem) = v.at(moveItem-1);
30         --moveItem;
31     }
32     v.at(moveItem) = insert;
33 }
34 ...
35 }
```

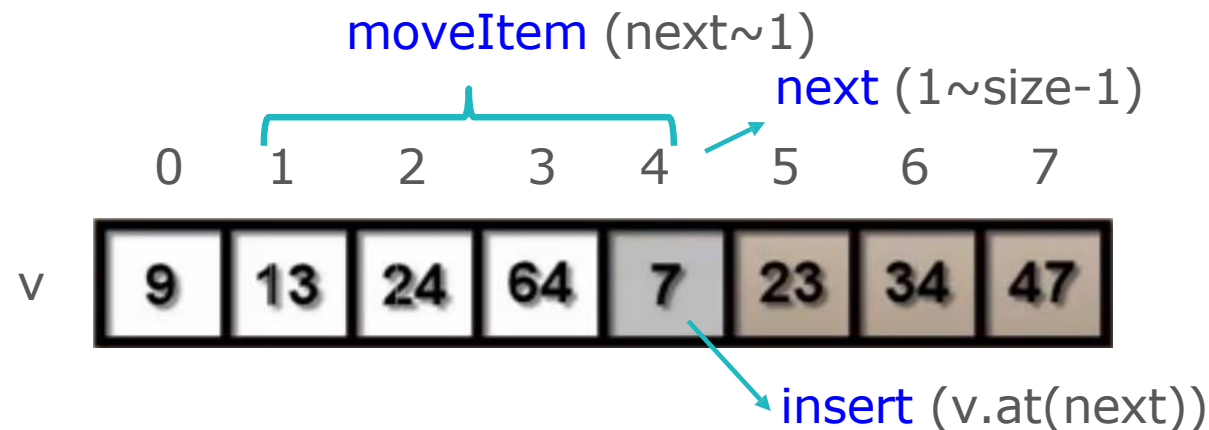
Output:

Unsorted array:

64 24 13 9 7 23 34 47

Step-by-step:

24	64	13	9	7	23	34	47
13	24	64	9	7	23	34	47
9	13	24	64	7	23	34	47
7	9	13	24	64	23	34	47
7	9	13	23	24	64	34	47
7	9	13	23	24	34	64	47
7	9	13	23	24	34	47	64



Using sort() in C++ Standard Library

sort(): 0.0547
seconds
v1 and v2 are
different.
insertion_sort():
154.26 seconds
v1 and v2 are
the same.

```
1 #include <vector>
2 #include <algorithm>
3 #include <iostream>
4 #include <cstdlib>
5 #include "Clock.h"
6 using namespace std;
7 void insertion_sort(vector<int> & v)
8 {
9     int insert, moveItem;
10    for(int next=1;next<v.size();++next)
11    {
12        insert = v.at(next);
13        moveItem = next;
14        while((moveItem>0) && (v.at(moveItem-1)
> insert))
15        {
16            v.at(moveItem) = v.at(moveItem-1);
17            --moveItem;
18        }
19        v.at(moveItem) = insert;
20    }
21 }
```

```
23 int main()
24 {
25     Clock clk;
26     const int size = 100000;
27     vector<int> v1(size),v2;
28     srand(time(NULL));
29     for(int i=0; i<size; ++i)
30         v1.at(i) = random();
31     v2 = v1;    clk.start();
32     sort(v1.begin(), v1.end());
33     cout << "sort(): " <<
clk.getElapsedTime()<< " seconds\n";
34     cout << "v1/v2 are " << ((v1==v2)?"the
same.\n":"different.\n");
35     clk.start();
36     insertion_sort(v2);
37     cout << "insertion_sort(): " <<
clk.getElapsedTime() << " seconds\n";
38     cout << "v1/v2 are " << ((v1==v2)?"the
same.\n":"different.\n");
39     return 0;
40 }
```

Clock.h and Clock.cpp

Clock.h

```
1 #include <ctime>
2 using namespace std;
3 class Clock {
4     public:
5         Clock();
6         Clock(clock_t s);
7         void start();
8         void setStart(clock_t start_ts);
9         clock_t getStart();
10        double getElapsedTime();
11    private:
12        clock_t start_ts;
13 };
```

Clock.cpp

```
1 #include "Clock.h"
2 Clock::Clock() { setStart(0); }
3 Clock::Clock(clock_t s) {
4     setStart(s);
5 }
6 void Clock::start() {
7     setStart(clock());
8 }
9 void Clock::setStart(clock_t ts) {
10     start_ts = (ts>0)?ts:clock();
11 }
12 clock_t Clock::getStart() {
13     return start_ts;
14 }
15 double Clock::getElapsedTime() {
16     return static_cast<double>(clock()-getStart())
17                                     / CLOCKS_PER_SEC ;
17 }
```

Reference

- Insertion Sort Concept,
<http://www.youtube.com/watch?v=Fr0SmtN0IJM&t=126>
- Insertion Sort Example,
<http://www.youtube.com/watch?v=c4BRHC7kTaQ&t=75>
- Insertion Sort with Romanian Folk Dance,
<http://www.youtube.com/watch?v=ROaIU379I3U>