数据结构
Data Structures

Chapter 2 Review of C++ Programming

Prof. Yitian Shao
School of Computer Science and Technology

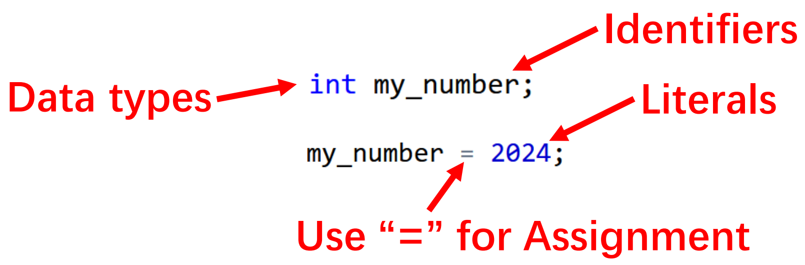
Review of C++ Syntax and Programming

Course Overview

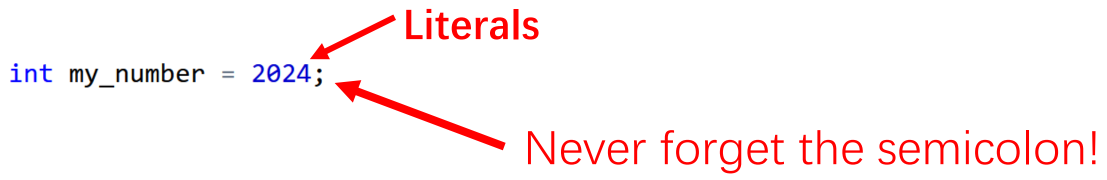
- Variable definition and assignment
- Use of arithmetic, relational, and logical operators
- Use of functions
- Use of string
- Use of array and vector
- Sequence, selection, and iteration code blocks
- Use of structures and classes
- Basics of input and output

Variable definition and assignment

- Initialize a variable



```
int my_number;
my_number = 2024;
```



```
int my_number = 2024;
```

Atomic Data types we use throughout the course:
int, float, char, string, bool

Operators

- Arithmetic operation
  - Increment (a = a + 1)

    a++ ++a
  - Decrement (a = a – 1)

    a-- --a

- Compound assignment

Compound operator	Equivalent arithmetic operation
a += b	a = a + b
a -= b	a = a - b
a *= b	a = a * b
a /= b	a = a / b
a %= b	a = a % b

Operator	Description
+a	unary plus
-a	unary minus
a*b	multiplication
a/b	division
a%b	modulus
a+b	addition
a-b	subtraction
=	direct assignment

Operators

- Relational operation

operand1 **relational_operator** *operand2*

Example: a > b

expression1 **relational_operator** *expression2*

Example: (a + b) > (a < b)

Relational Operator	Meaning
>	Greater than
<	Less than
>=	Greater than equal to
<=	Less than equal to
==	Equal to
!=	Not equal to

- Logical operation

Logical operators	Description
&&	AND: **true** only if both operands are **true**
||	OR: **true** if any of the operands is **true**
!	NOT: reverse the logic

Functions

```
type functionName (type name, type name, …, type name)
{
        statement;
        statement;
        …
        statement;
        return expression; // if return type is not void
}
```

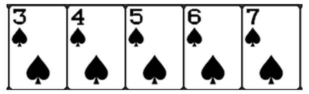
Calling the function:

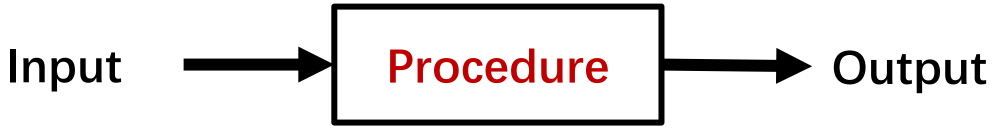
```
functionName (value, value, …, value);
```

Basic Concept of Algorithms

- An **algorithm** describes **how to solve a problem**; it is a **procedure** that takes in **input**, follows a certain set of steps, and then produces an **output**

Example problem: Given a set of five cards (randomly shuffled), pick the **largest one**





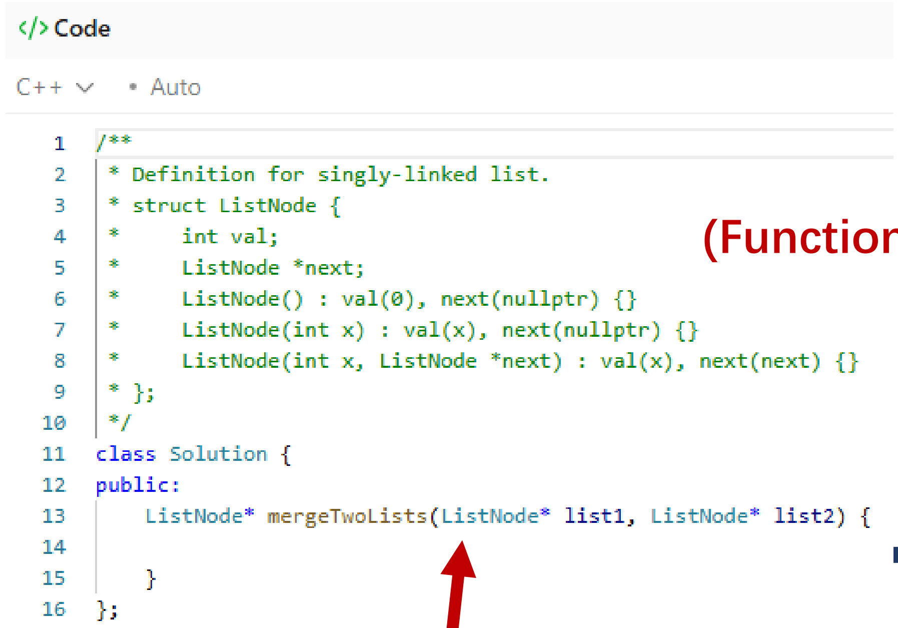
Input: A set of 5 cards

Output: The card with the largest value

Procedure: **You will learn how to implement it in this course**

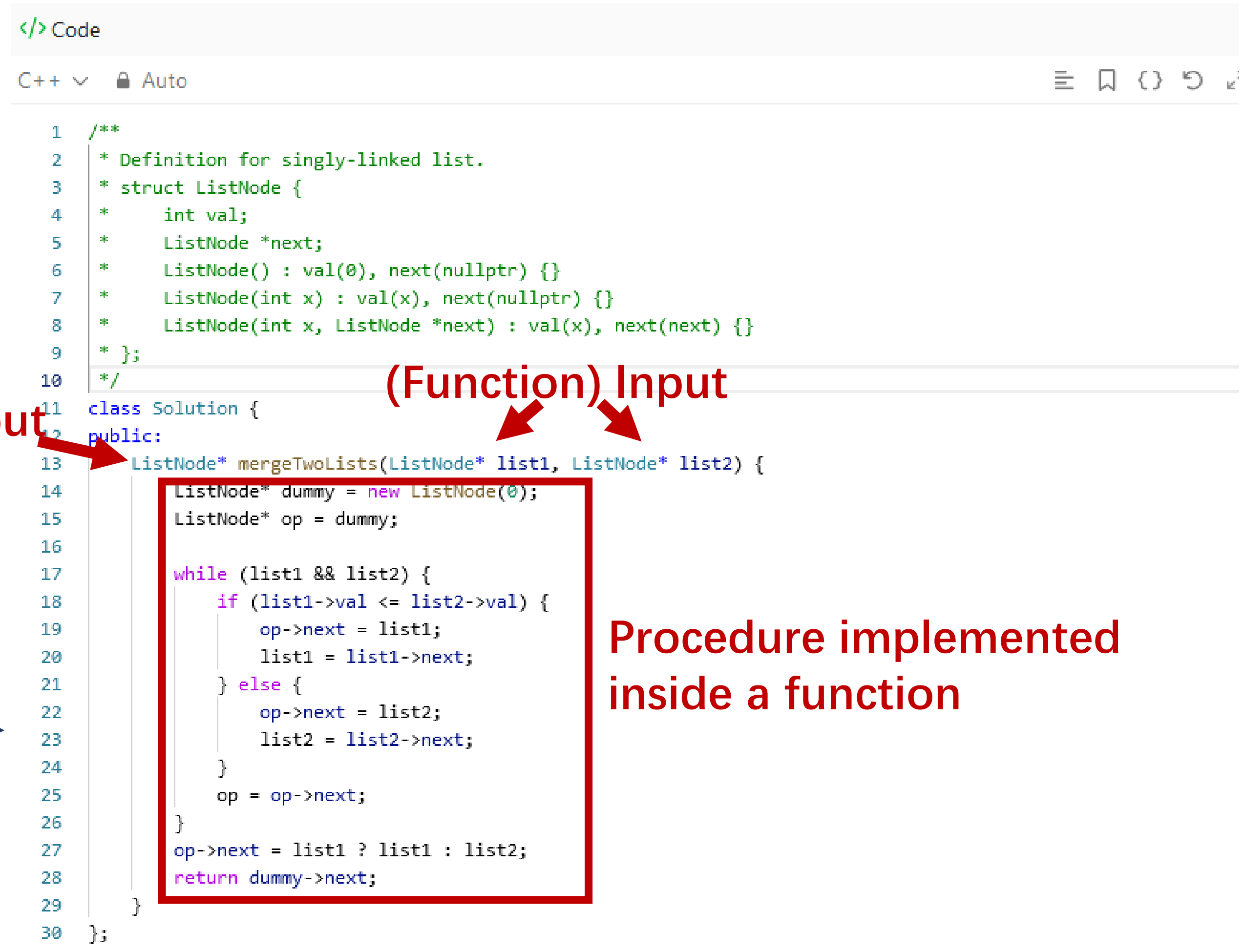
How to evaluate your implemented algorithm

$$\begin{tikzcd}\text{Input} \arrow[r] & \boxed{\text{Procedure}} \arrow[r] & \text{Output}\end{tikzcd}$$



```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {

    }
};
```

Throughout the course, you only need to focus on filling out the content of a given **Function**



```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode* dummy = new ListNode(0);
        ListNode* op = dummy;

        while (list1 && list2) {
            if (list1->val <= list2->val) {
                op->next = list1;
                list1 = list1->next;
            } else {
                op->next = list2;
                list2 = list2->next;
            }
            op = op->next;
        }
        op->next = list1 ? list1 : list2;
        return dummy->next;
    }
};
```

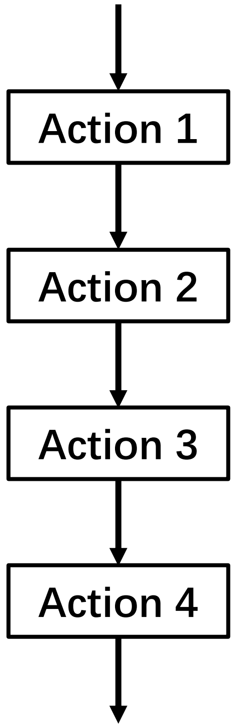
(Function) Output

(Function) Input

Procedure implemented inside a function

Building blocks of algorithms

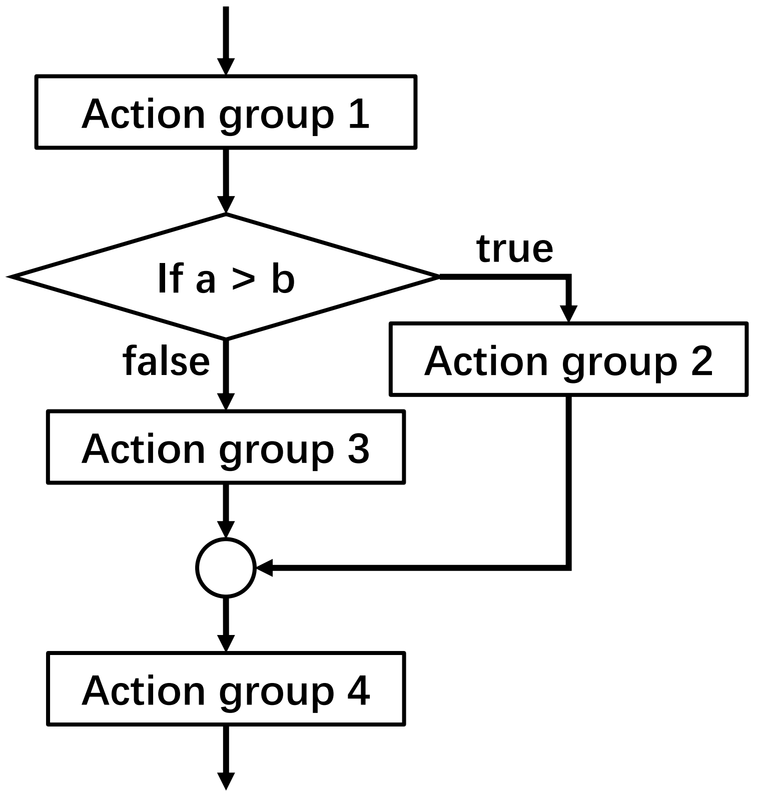
- Sequence
- Selection
- Iteration



Building blocks of algorithms

- Sequence
- Selection
- Iteration

```
// Action group 1
if(a > b)
{
    // Action group 2
}
else
{
    // Action group 3
}
// Action group 4
```



```
switch( expression )
{
        case  value_1:
                {Action group 1}
                break;

        case  value_2:
                {Action group 2}
                break;

                ⋮

        default:
                {Action group n}
                break;
}
```

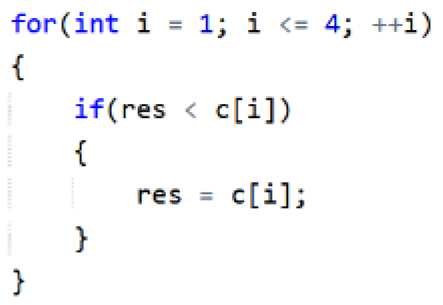
Building blocks of algorithms

- Sequence
- Selection
- Iteration

Three types of loops in C++

- **for** loop

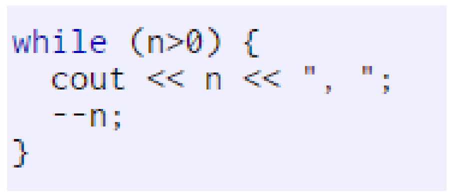
```
for (initialization ; condition ; update )
{
        Actions
}
```



```
for(int i = 1; i <= 4; ++i)
{
    if(res < c[i])
    {
        res = c[i];
    }
}
```

- **while** loop

```
while (condition)
{
        Actions
}
```



```
while (n>0) {
  cout << n << ", ";
  --n;
}
```

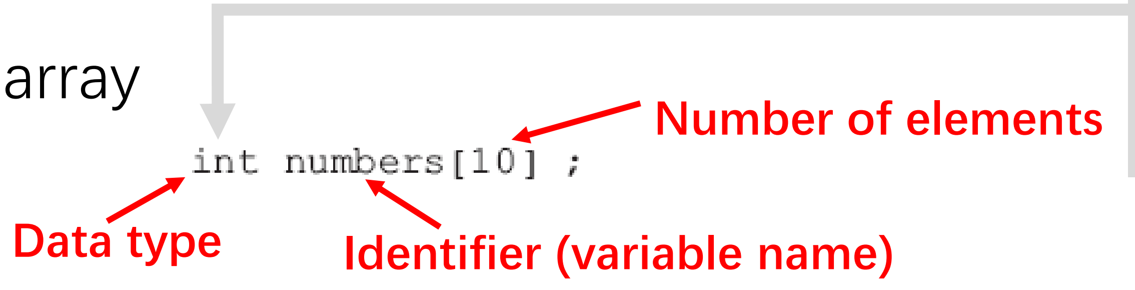
- **do-while** loop

```
do
{
        Actions
}
while (condition)
```

Array

- A group of variables of the **same data type**

- Define an array



```
int numbers[10] ;
```

Data type of an array can be:
int
float
char
User-Defined Data Types (UDTs)

- **Size of the array is fixed**: the **number of elements** in an array

- Array indexing: **index** decide the **position** of an element in an array

variable_name [index]

Vector (dynamic size array)

- In C++, both **array** and **vector** are used to store collections of data. **Array** has a fixed size while **vector** has **dynamic size**: can change during runtime.
- **vector** is part of the C++ Standard Template Library (**STL**)
- Syntax of Vector

vector<data_type> vec_name;

```
#include <iostream>
#include <vector>
using namespace std;

int main() {  // creating a vector of integers
  vector<int> vec = { 1, 2, 3, 4, 5 };
  vec.push_back(6); // Add an element to the end  {1, 2, 3, 4, 5, 6}
  vec.pop_back(); // Remove the last element   {1, 2, 3, 4, 5}
 vec.erase(vec.begin()); // Delete the first element  {2, 3, 4, 5}

  return 0;
}
```

String

- The string data type is not built into C++ and is a class defined in **C++ STL**

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string greeting = "Hello, World!"; // Creating and initializing strings
    cout << greeting << endl;

    int length = greeting.length();

    char firstChar = greeting[0];
    char secondChar = greeting.at(1);

    return 0;
}
```

Pointer and Reference

	Pointer	Reference

Declare	int* myPtr1
	Float* myPtr2
	char* myPtr3
		void myFunction(int& myVariable){}
		void swap_vals(float& val1, float& val2){}

Access	*myPtr1
	*myPtr2
	*myPtr3
		int x; float y;
		myPtr1 = &x;
		myPtr2 = &y;

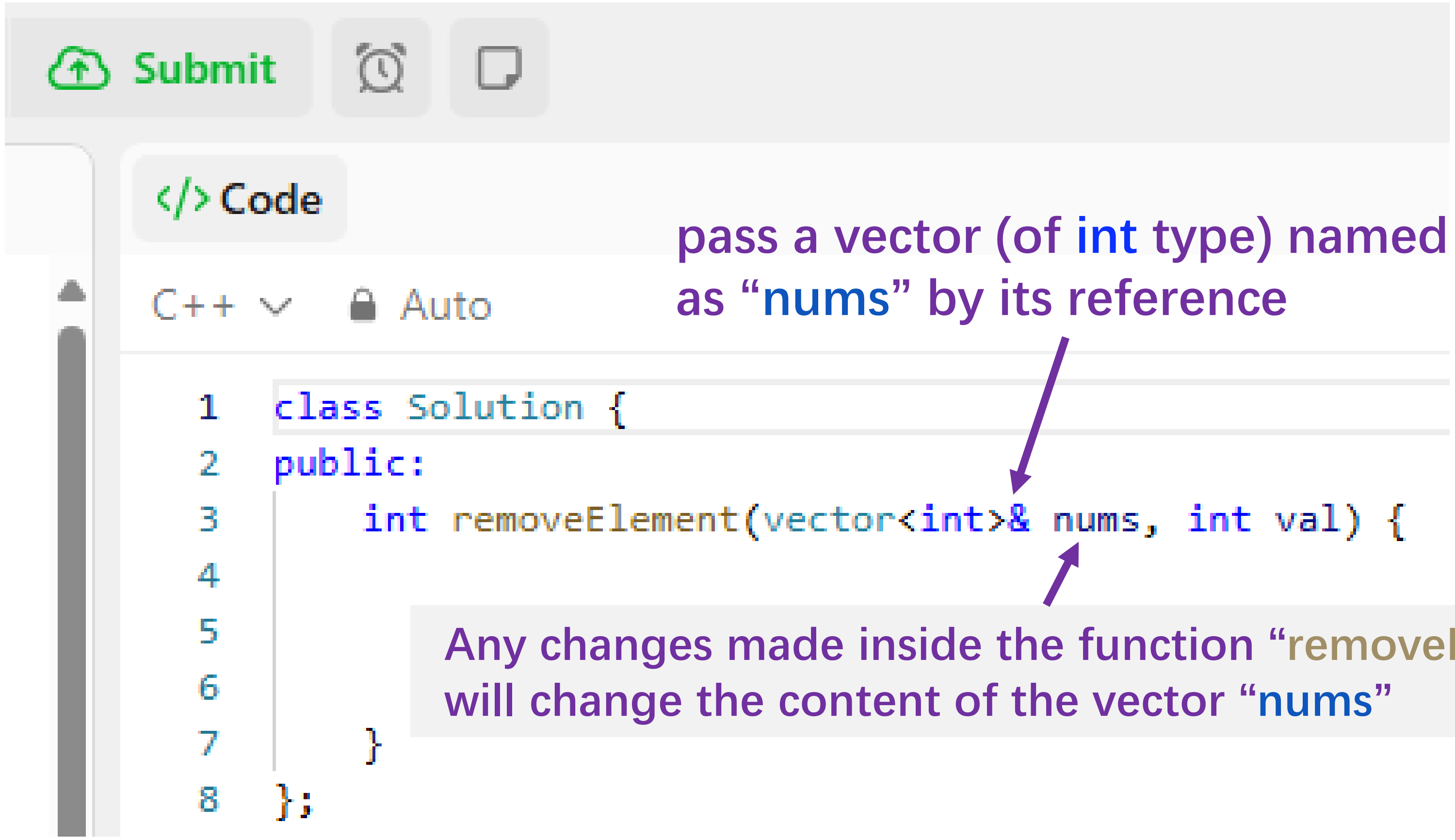
pass by reference

dereferencing

getting address

Passing Reference

Example of passing variable by reference to a function
LeetCode 27 will be assigned as homework exercise



```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {


    }
};
```

pass a vector (of int type) named as “nums” by its reference

Any changes made inside the function “removeElement” will change the content of the vector “nums”

Structure

A structure template consists of the reserved keyword **struct** followed by the **name of the structure**

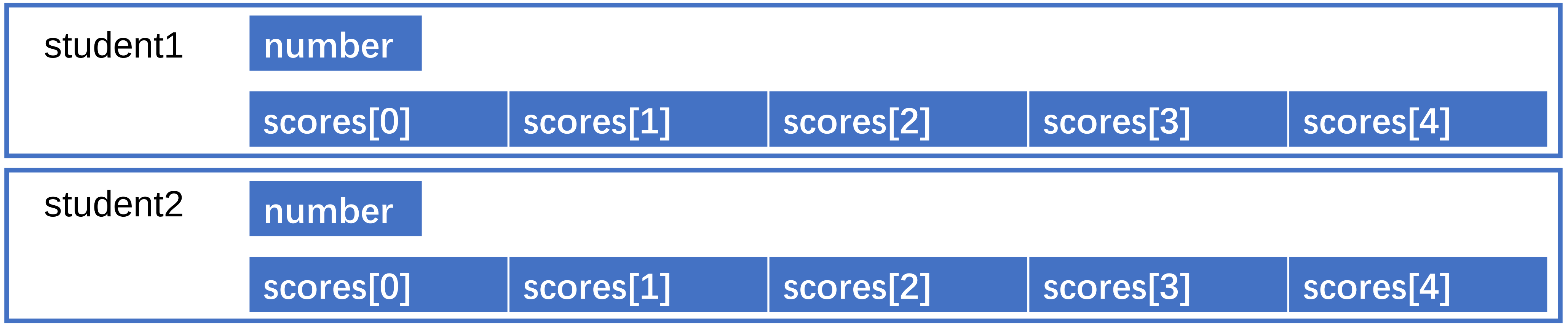
```
struct student_rec
{
  int number ;    // Student number.
  float scores[5] ;   // Scores on five tests.
} ;
```

name of the structure

structure member: each item in the structure

Define **student1** and **student2** to be of the type **struct** student_rec

```
struct student_rec student1, student2 ;
```



Class and Object-Oriented Programming

Class

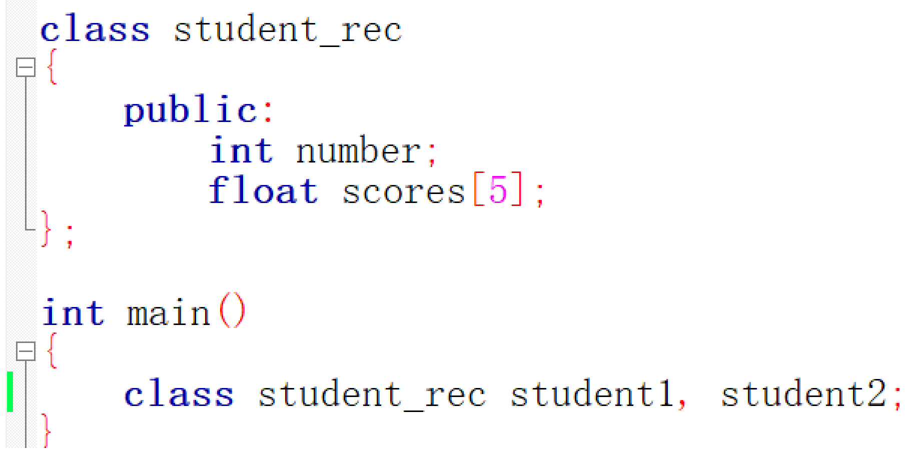
Members of a class are **private** by default.

Declared using the **class** keyword.

Normally used for **data abstraction** and **inheritance**.

Syntax:

```
class class_name {
    data_member;
    member_function;
};
```



```
class student_rec
{
    public:
        int number;
        float scores[5];
};

int main()
{
    class student_rec student1, student2;
}
```

C++ Structure

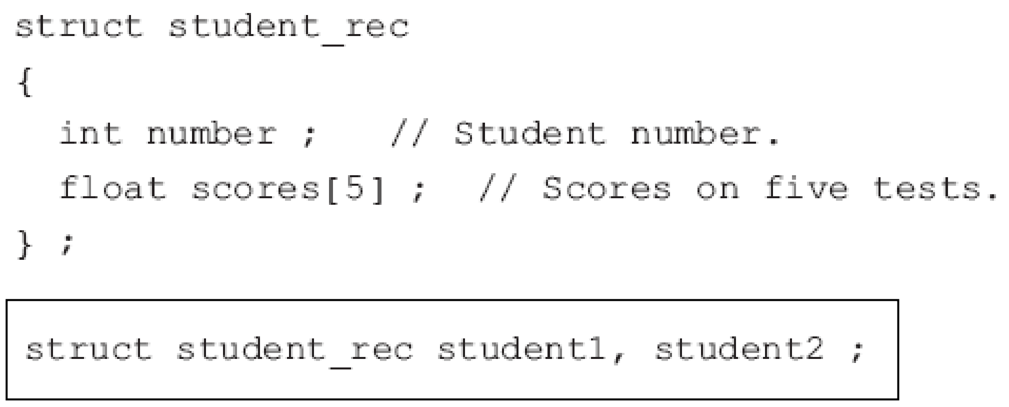
Members of a structure are **public** by default.

Declared using the **struct** keyword.

Normally used for the **grouping of different datatypes**.

Syntax:

```
struct structure_name {
    structure_data_member;
    structure_member_function;
};
```



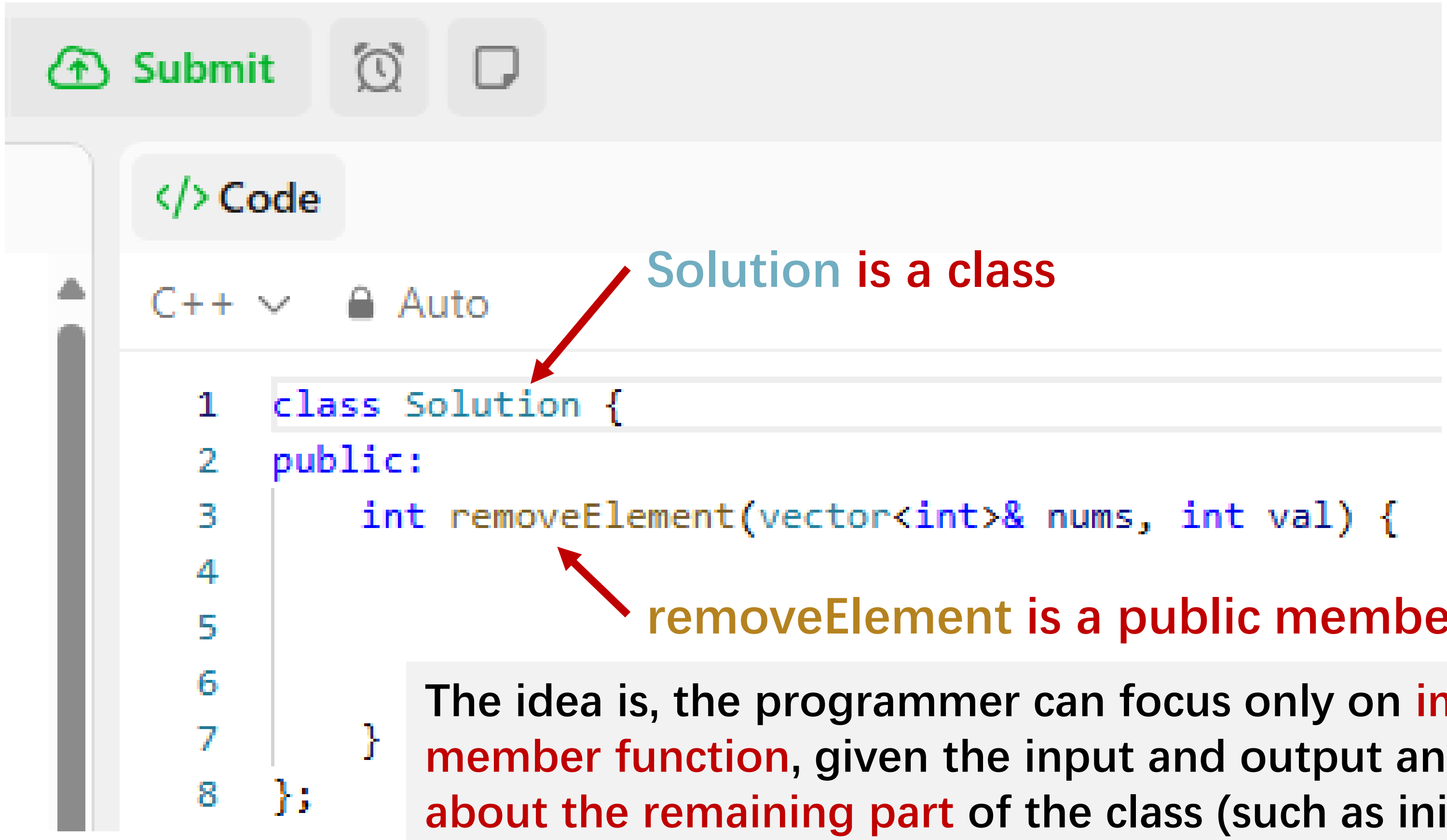
```
struct student_rec
{
  int number ;    // Student number.
  float scores[5] ;  // Scores on five tests.
} ;
```

```
struct student_rec student1, student2 ;
```

Class and Object-Oriented Programming

Example of Class

LeetCode 27 will be assigned as homework exercise



```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {


    }
};
```

Solution is a class

removeElement is a public member function of the class

The idea is, the programmer can focus only on implementing the member function, given the input and output and need not to worry about the remaining part of the class (such as initializing the inputs and testing the implemented function)

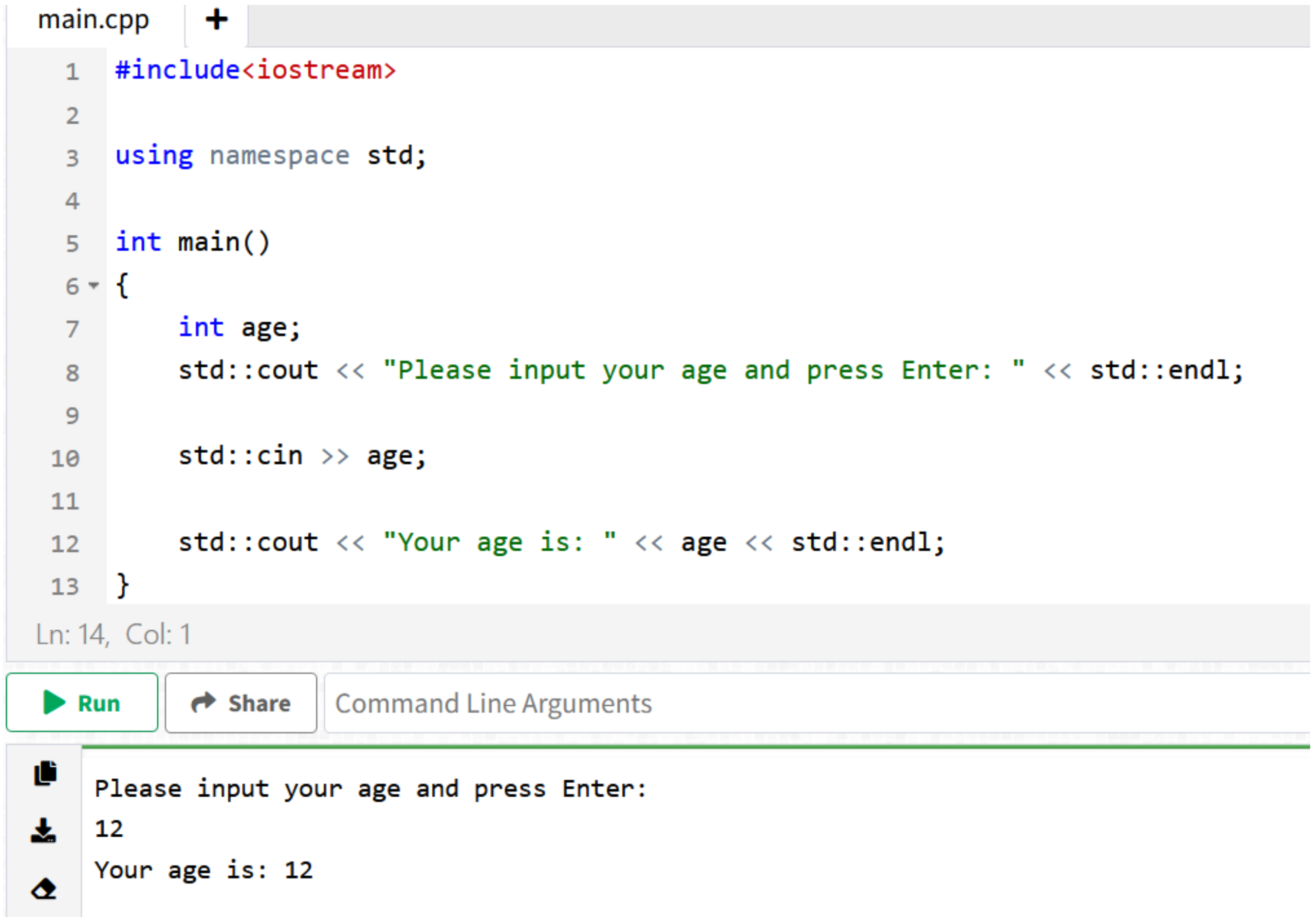
Input and Output

- Output
  std::cout << *a_string_constant*
  std::cout << *a_variable*

  - Concatenate multiple outputs

  - Change line:
  std::endl

- Input
  std::cin >> *a_variable*

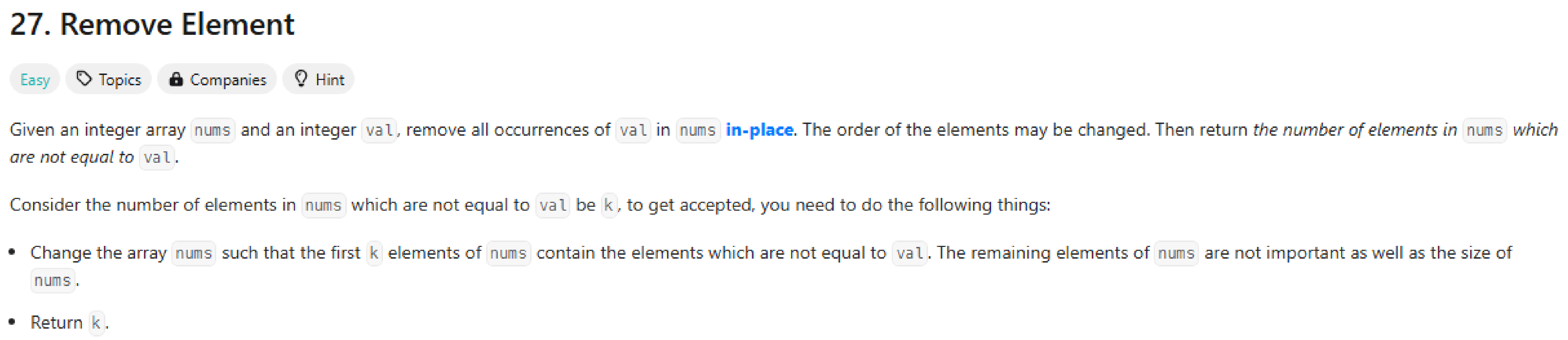


```cpp
#include<iostream>

using namespace std;

int main()
{
    int age;
    std::cout << "Please input your age and press Enter: " << std::endl;

    std::cin >> age;

    std::cout << "Your age is: " << age << std::endl;
}
```

```
Please input your age and press Enter:
12
Your age is: 12
```

Exercise 2.1

- Complete LeetCode 27



27. Remove Element

Easy | Topics | Companies | Hint

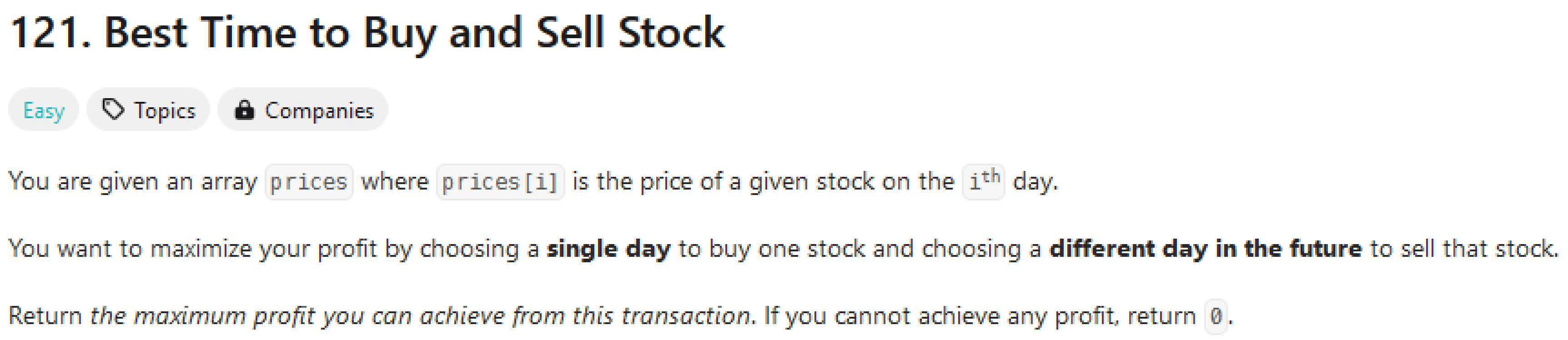
Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. The order of the elements may be changed. Then return *the number of elements in* `nums` *which are not equal to* `val`.

Consider the number of elements in `nums` which are not equal to `val` be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Exercise 2.2

- Complete LeetCode 121



121. Best Time to Buy and Sell Stock

Easy | Topics | Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the `i`th day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.