Programming in C++

Tony Wong (microtony)
2021-02-20

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Why C++?

- C++ shares similar syntax with many other programming languages
  - Java, Javascript, C#, Objective C, PHP etc..
- Centered around important CS concepts
  - Data types, Control structures, Object-oriented programming
- Wide range of applications and can be run in different environments
  - Servers, operating systems, games, embedded systems, etc
  - C++ standard provides a portable interface. Programs can be compiled into executable for different systems.

Why C++ for competitive programming?

- C++ programs runs the fastest
  - In many contests, problems are not guaranteed to be solvable by all languages.
    IOI: "The ISC and ITC do not want to put a guaranteed percentage on points that could be gained by a second class language, or have different time limits based on language."
    Code Jam: "... it is not guaranteed that any problem can be solved in any language; ... Just as in everyday software engineering, part of the contest is using the right tool for each job!"
- C++ STL comes with useful algorithms and data structures
  - Sorting, binary search, stack, heap, BST, etc...
- C++ programs are easy to debug
  - Compilation step can help uncover bugs

```python
def has_odd(l):
  flag = False
  for elm in l:
    if elm % 2 == 1:
      flga = True
  return flag
```

This Python program would not even cause runtime error

Today I am going to teach you how to write C++ programs.

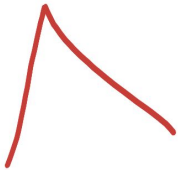
Survey

How well do you know C++?

- I don't know C++ at all
- I know about the basics and can write simple problems
- I know a lot about the language features, enough for me to solve OI problems.
- I am an expert in C++. I have knowledge beyond those needed in OI.

Today I am going to teach you how to write C++ programs.

Today I am going to teach you how to write C++ programs.



Therefore, we will focus on making the best use of new C++ features.

Unless otherwise stated, we are using **C++17** here.

To learn more about data types and C++ language, you may refer to the 2019 slides.

Introduction to C++ Data Processing

Basic program structure

```cpp
#include <iostream>
using namespace std;
int main() {
  cout << "Hello, World!" << endl;
  return 0;
}
```

Include library headers

`<iostream>` provides input and output functionality (`cout` and `endl` in this example)

#include <iostream>

Basic program structure

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  cout << "Hello, World!" << endl;
  return 0;
}
```

When using GCC C++ compiler, `<bits/stdc++.h>` provides most functions needed for competitive programming

- Shorter header
- Avoid compilation errors caused by missing header, especially in contests with no feedback

<bits/stdc++.h>

Basic program structure

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  cout << "Hello, World!" << endl;
  return 0;
}
```

This line is to “move” everything in the `std` namespace into the our program.

Pro:

- No need to type `std::` prefix

Con:

- Program may not be forward compatible

namespace

Basic program structure

```cpp
#include <bits/stdc++.h>

int main() {
  std::cout << "Hello, World!" <<
      std::endl;
  return 0;
}
```

Example without
`using namespace std;`

Basic program structure

```
#include <bits/stdc++.h>
using std::cout;
using std::endl;
int main() {
  cout << "Hello, World!" << endl;
  return 0;
}
```

Bring in specific symbols.

This won't have the forward compatibility issue.

using declaration

Forward compatibility issue

```cpp
#include <bits/stdc++.h>
using namespace std;
// Move first character to the end.
string move(string s) {
  return s.substr(1) + s[0];
}
int main() {
  cout << move("abcdef") << endl;
  return 0;
}
```

What does the program output when compiled in C++03?

```
g++ -std=c++03 program.cpp -o program
```

What does the program output when compiled in C++11?

```
g++ -std=c++11 program.cpp -o program
```


substr

Basic program structure

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  cout << "Hello, World!" << endl;
  return 0;
}
```

This is the main program.

Note that the return type is `int`.

Basic program structure

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  cout << "Hello, World!" << endl;
  return 0;
}
```

```
(windows) a.exe && b.exe
(linux)   ./a && ./b
```

A return code of **0** indicates that the program ended successfully.

Other numbers can be used to indicate that there is some warning / error.

You can use && in the console to chain commands. In this example, program B runs only if program A returns 0.

`return 0;` is optional

Main function

Basic program structure

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  cout << "Hello, World!" << endl;
  return 0;
}
```

Send “Hello, World!” and line break to the output stream.

`endl` also flushes the stream. (Useful for interactive tasks).

Arrays

We can use a character array to store strings.

```cpp
#include <bits/stdc++.h>
using namespace std;
int a[10];
const char s[] = "HKOI";
int main() {
  cin >> a[0] >> a[1];
  cout << a[0] + a[1] << endl;
  cout << s << endl;
  return 0;
}
```

Input

```
4 7
```

Output

```
11
HKOI
```

C++ array

With the exception of const arrays, (e.g. `const char s[]`) modern C++ discourages the use of raw arrays.

The type and size of an array is fixed once declared.

```cpp
#include <bits/stdc++.h>
using namespace std;
array<int, 10> a;
const char s[] = "HKOI";
int main() {
  cin >> a[0] >> a[1];
  cout << a[0] + a[1] << endl;
  cout << s << endl;
  return 0;
}
```

Input

```
4 7
```

Output

```
11
HKOI
```

C++ array

Arrays can also be declared with initialization.

The size and type will be automatically determined.

Here, the type of a is `array<int, 3>`.

```cpp
#include <bits/stdc++.h>
using namespace std;
array a{4, 8, 3};
int main() {
  cout << a[0] + a[1] + a[2] << endl;
  return 0;
}
```

Input

Output

15

Benefits of C++ array

For C array, the identifier degenerates into a pointer when passed into functions.

Provides index checking via `.at(index)`, which makes debugging easier.

```cpp
#include <bits/stdc++.h>
using namespace std;
int a[] = {4, 8, 3};
int main() {
  cout << a[0] + a[3] << endl;
  return 0;
}
```

Likely Output: 4

```cpp
#include <bits/stdc++.h>
using namespace std;
array a{4, 8, 3};
int main() {
  cout << a.at(0) + a.at(3) << endl;
  return 0;
}
```

Runtime error

```
terminate called after throwing an instance of 'std::out_of_range'
  what():  array::at: __n (which is 3) >= _Nm (which is 3)
```

at

Dynamic size array: vector

Very often the task requires us to read N integers.

We can use `vector`, which is a dynamic size array to store the data.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; ++i) {
    cin >> a[i];
  }
  int sum = 0;
  for (int i = 0; i < n; ++i) {
    sum += a[i];
  }
  cout << sum << endl;
  return 0;
}
```

Initial size

Input

```
6
1 4 2 8 5 7
```

Output

```
27
```

vector

Dynamic size array: vector

Alternatively, we can start with an empty vector and use `push_back(x)` to add items to the vector while reading.

```cpp
int main() {
  int n;
  cin >> n;
  vector<int> a;
  for (int i = 0; i < n; ++i) {
    int x;
    cin >> x;
    a.push_back(x);
  }
  int sum = 0;
  for (int i = 0; i < n; ++i) {
    sum += a[i];
  }
  cout << sum << endl;
  return 0;
}
```


```
Input

6
1 4 2 8 5 7
```

```
Output

27
```

[push_back]

vector assignment

We can replace the entire vector by assigning another vector to it.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  vector<int> a(4, 10);
  cout << a[0] << endl;
  a = vector<int>{1, 2, 3};
  cout << a[1] << endl;
  return 0;
}
```

```
Output

10
2
```


operator=

Iterate over vector - int i

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  for (int i = 0; i < a.size(); ++i) {
    cout << a[i] << endl;
  }
  return 0;
}
```

Current size

```
Output

1
4
2
8
5
7
```

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  for (int i = 0; i + 1 < a.size(); ++i) {
    cout << a[i] - a[i + 1] << endl;
  }
  return 0;
}
```

```
Output

-3
2
-6
3
-2
```

Is it ok to write **`i < a.size() - 1`**?

size

Iterate over vector - range-based loop

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  for (int x : a) {
    cout << x << endl;
  }
  return 0;
}
```

```
Output

1
4
2
8
5
7
```

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  for (auto x : a) {
    cout << x << endl;
  }
  return 0;
}
```

```
Output

1
4
2
8
5
7
```

You can use **auto** when type can be automatically determined.

Range-based loop

Modifying values in range-based loop

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  for (int x : a) {
    x = 3;
  }
  cout << a[0] << endl;
  return 0;
}
```

Value of v[0], v[1], ... is copied to x

```
Output

1
```

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  for (int& x : a) {
    x = 3;
  }
  cout << a[0] << endl;
  return 0;
}
```

Reference: x is same as a[0], a[1]...

```
Output

3
```


Reference declaration

Iterate over vector - iterator

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  for (auto it = a.begin(); it != a.end(); ++it) {
    cout << *it << endl;
  }
  return 0;
}
```

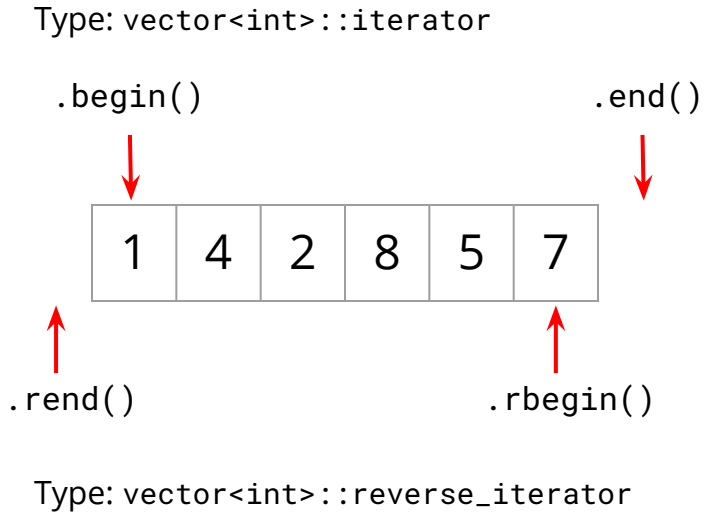
Advance the iterator

De-reference (get data being pointed at)

```
Output

1
4
2
8
5
7
```

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  for (auto it = a.rbegin(); it != a.rend(); ++it) {
    cout << *it << endl;
  }
  return 0;
}
```

```
Output

7
5
8
2
4
1
```



香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

begin end rbegin rend
RandomAccessIterator

Modifying values using iterator

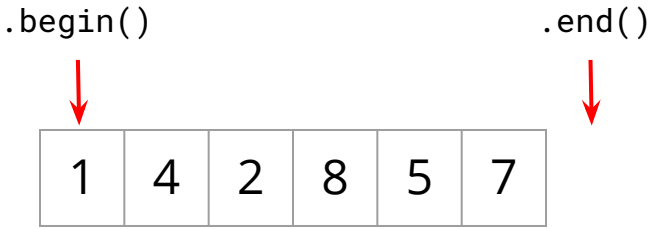
```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  auto it = a.begin();
  *it = 3;
  cout << a[0] << endl;
  return 0;
}
```

```
Output

3
```

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  auto it = a.end();
  *it = 3;
  cout << a[0] << endl;
  return 0;
}
```

```
Output

1
```

Possibly runtime error



Find element in vector

`find(first, last, value)` searches for `value` between `[first, last)` and returns an iterator to the first matching `value`.

It returns `last` if `value` is not found.

Use `distance` to get the index.

```cpp
#include <bits/stdc++.h>
using namespace std;
vector<int> a{1, 4, 2, 8, 5, 7};
int main() {
  auto it1 = find(a.begin(), a.end(), 8);
  cout << distance(a.begin(), it1) << endl;
  auto it2 = find(a.begin(), a.end(), 3);
  cout << (it2 == a.end()) << endl;
  return 0;
}
```

```
Output

3
1
```


find distance

Sort a vector

Use `sort(first, last)` to sort a vector in ascending order.

Use `reverse(first, last)` to reverse a vector.

```
vector<int> a{1, 4, 2, 8, 5, 7};
sort(a.begin(), a.end());
for (int x : a) {
  cout << x << " ";
}
cout << endl;
reverse(a.begin(), a.end());
for (int x : a) {
  cout << x << " ";
}
cout << endl;
```

```
Output

1 2 4 5 7 8
8 7 5 4 2 1
```

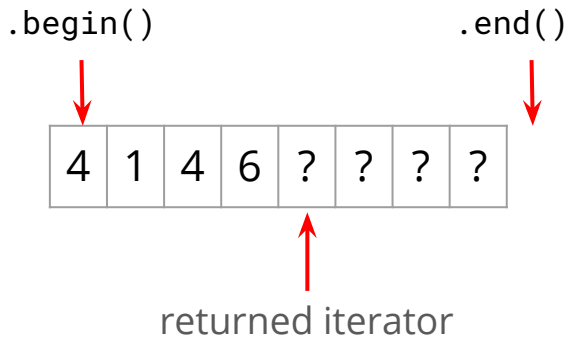

sort reverse

Removing duplicates from a vector

Use `unique(first, last)` to move the first element in each identical group to the front, keeping their relative order. The returned iterator points to the position after the last remaining element.

Use `erase(first, last)` to remove elements from a vector.

```
vector<int> a{4, 4, 1, 1, 1, 4, 6, 6};
auto it = unique(a.begin(), a.end());
for (int x : a) {
  cout << x << " ";
}
cout << endl;
a.erase(it, a.end());
for (int x : a) {
  cout << x << " ";
}
cout << endl;
```




```
Output

4 1 4 6 1 4 6 6
4 1 4 6
```


unique erase

Comparing vectors

You can use equality operator `==` `!=` to check if the vectors has the same length and the same contents in the same order.

Comparison operators `<` `<=` `>=` `>` compare 2 vectors in lexicographical order.

```
vector<int> a{3, 4, 5};
vector<int> b{3, 4};
vector<int> c{3, 4, 6};
vector<int> d{3, 4, 5};
cout << (a == b) << endl;
cout << (a > b) << endl;
cout << (a > c) << endl;
cout << (a == d) << endl;
```

```
Output

0
1
0
1
```

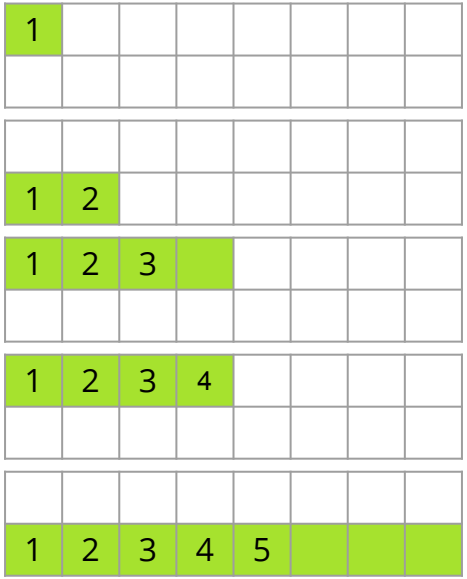
vector comparison

Internal storage

Vector always store the data in contiguous segment of memory.

When it is already full and you try to push one more element, it finds a larger piece of memory elsewhere and move all the data there.

```
vector<int> a;
cout << a.capacity() << " ";
cout << a.data() << endl;
for (int i = 1; i <= 6; ++i) {
  a.push_back(i);
  cout << a.capacity() << " ";
  cout << a.data() << endl;
}
```




```
Output

0 0
1 0x192400
2 0x196230
4 0x192400
4 0x192400
8 0x196230
8 0x196230
```

capacity data

Time complexity for push_back

Assume that you call `push_back(x)` N times.

The total cost comprises of:

- Cost of adding an element
  - 1 operation per push_back
  - Total N operations for N push_back
- Cost of moving elements when vector is full
  - 1, 2, 4, 8, ..., 2^k (where 2^k < N)
  - The sum of above = 2^(k+1) - 1 < 2N

Total cost for N push_back = N + (<2N) < 3N, and therefore is O(N)
We can say that push_back is amortized O(1)

push_back

Be careful about iterators

Some manipulation operations, especially when they affect the internal storage, **invalidate** iterators. Read the docs for details.

If unsure, always get fresh iterators.

```
vector<int> a{1, 2, 3};
auto it = a.begin();
cout << a.capacity() << " " << *it << endl;
a.push_back(4);  // it is invalidated.
cout << a.capacity() << " " << *it << endl;
```

```
Output

3 1
6 7890304
```

push_back

2D vector

```
vector<vector<int>> a{{1, 2, 3}, {4}, {5, 6}};
cout << a[0].size() << endl;
cout << a[1].size() << endl;
cout << a[2].size() << endl;
cout << a[2][0] << endl;
```

```
Output

3
1
2
5
```

```
int n = 4;
int m = 5;
vector<vector<int>> a(n, vector<int>(m));
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < m; ++j) {
    cin >> a[i][j];
  }
}
```

```
int n = 4;
int m = 5;
vector<vector<int>> a;
for (int i = 0; i < n; ++i) {
  vector<int> b;
  for (int j = 0; j < m; ++j) {
    int x;
    cin >> x;
    b.push_back(x);
  }
  a.push_back(b);
}
```

Remember move()?

```cpp
vector<vector<int>> a;
vector<int> b{1, 2, 3};
cout << b.data() << endl;
a.push_back(b);
cout << a[0].data() << endl;
cout << b.size() << endl;
```

```
Output

0x22400
0x26580
3
```

Data is copied

```cpp
vector<vector<int>> a;
vector<int> b{1, 2, 3};
cout << b.data() << endl;
a.push_back(move(b));
cout << a[0].data() << endl;
cout << b.size() << endl;
```

```
Output

0xf82400
0xf82400
0
```

Not guaranteed to be 0

`std::move()` is complicated.
No need to care about it in competitive programming.

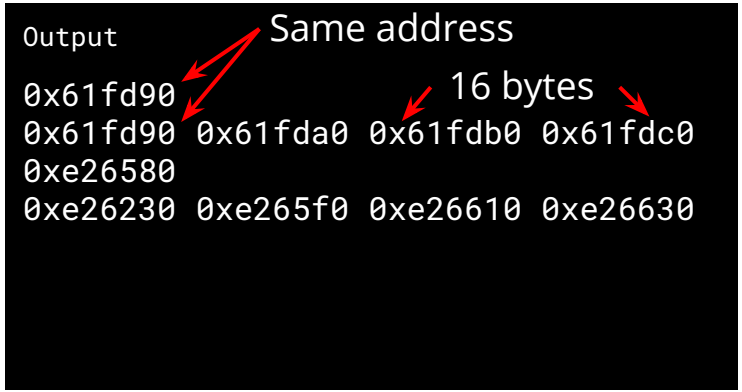
data move

Difference between array and vector

Using 2D array and 2D vector as an example

- All data in a 2D array are contiguous in memory
- Each row of data in a 2D vector can stored separately

```
array<array<int, 4>, 4> a;
vector<vector<int>> b(4, vector<int>(4));
cout << a.data() << endl;
cout << a[0].data() << " " << a[1].data() << " ";
cout << a[2].data() << " " << a[3].data() << endl;
cout << b.data() << endl;
cout << b[0].data() << " " << b[1].data() << " ";
cout << b[2].data() << " " << b[3].data() << endl;
```



vector<bool>

`vector<bool>` is a very special kind of vector.

Its implementation allows efficient storage of bools, 1 bit (vs 1 byte) for each bool.

Some vector functions cannot be used.

```
vector<bool> a{true, false, true};
cout << a[0] << a[1] << a[2] << endl;
cout << a.capacity() << endl;
a.flip();
cout << a[0] << a[1] << a[2] << endl;
```

```
Output

101
32
010
```

string

C++ strings are very easy to use.

You can concatenate strings together using the + operator.

```
string s = "ab";
string t = "d";
s += 'c';  // append a character
t += "ef";  // append a string
cout << s.length() << endl;
cout << s + t << endl;
```

```
Output

3
abcdef
```


length operator+ operator+=

Iterate over string

You can also use ranged-based loop to iterate over a string.

```
string s = "abcdef";
for (int i = 0; i < s.length(); ++i) {
  cout << s[i] << endl;
}
for (char& c : s) {
  c -= 32;
}
cout << s << endl;
```

```
Output

a
b
c
d
e
f
ABCDEF
```

Using string::iterator

You can get iterators from string to perform operations similar to vector.

```
string s = "abcdef";
for (auto it = s.begin(); it != s.end(); ++it) {
  cout << *it << endl;
}
reverse(s.begin(), s.end());
cout << s << endl;
```

```
Output

a
b
c
d
e
f
fedcba
```

Using string::iterator

You can get iterators from string to perform operations similar to vector.

```
string s = "abcdef";
for (auto it = s.begin(); it != s.end(); ++it) {
  cout << *it << endl;
}
reverse(s.begin(), s.end());
cout << s << endl;
```

```
Output

a
b
c
d
e
f
fedcba
```

string comparison

You can compare strings directly using comparison operators.

You can also use `.compare()`, which returns 0 when the strings are equal, negative number when the left string is smaller, and positive otherwise.

```
cout << (string{"abc"} == string{"abc"}) << endl;
cout << (string{"abc"} < string{"def"}) << endl;
cout << (string{"abcd"} > string{"abc"}) << endl;
cout << string{"abc"}.compare("abx") << endl;
cout << string{"xyz"}.compare("xyz") << endl;
cout << string{"def"}.compare("a") << endl;
```

```
Output

1
1
1
-1
0
1
```

It can be any negative integer

It can be any positive integer

compare

Break

Please read M2102 problem statement

Functions

Functions with return type should have a return statement.

Void functions can omit the return statement.

```
int Square(int x) {
  return x * x;
}
void PrintMax(int a, int b, int c) {
  cout << max(a, max(b, c)) << endl;
}
int main() {
  cout << Square(5) << endl;
  PrintMax(4, 9, 1);
  return 0;
}
```

```
Output

25
9
```


Function declaration

Early Return

If you have a return statement at the end of a if block, no need to add else.

```
void PrintMax(int a, int b, int c) {
  if (a > b && a > c) {
    cout << a << endl;
    return;
  }
  cout << (b > c ? b : c) << endl;
}
int main() {
  PrintMax(4, 9, 1);
  return 0;
}
```



```
Output

9
```

Pass by reference

Pass by reference makes the identifier refer to the same variable specified in the argument. Therefore, the value can be changed inside the function.

```
void PassByValue(int a) {
  a = 5;
}
void PassByReference(int& a) {
  a = 5;
}
int main() {
  int x = 1;
  int y = 2;
  PassByValue(x);
  PassByReference(y);
  cout << x << " " << y << endl;
  return 0;
}
```

Value 1

Refers to y

```
Output

1 5
```

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Reference declaration

Pass by reference: vector

All types are passed by value. (unlike Java / Javascript)

```
void PassByValue(vector<int> a) {
  a[0] = 5;
}
void PassByReference(vector<int>& a) {
  a[0] = 5;
}
int main() {
  vector<int> x{1};
  vector<int> y{2};
  PassByValue(x);
  PassByReference(y);
  cout << x[0] << " " << y[0] << endl;
  return 0;
}
```

Value {1}

Refers to y

```
Output
1 5
```

Pass by value is slow

```cpp
int PassByValue(string s) {
  return s.length();
}
int PassByReference(string& s) {
  return s.length();
}
int main() {
  auto start_time = chrono::steady_clock::now();
  string s = "abcdefghijklmnopqrstuvwxyz";
  int total = 0;
  for (int i = 0; i < 10000000; ++i) {
    total += PassByValue(s);
  }
  cout << total << endl;
  auto end_time = chrono::steady_clock::now();
  cout << chrono::duration<double>(end_time - start_time).count() << endl;
  start_time = end_time;
  total = 0;
  for (int i = 0; i < 10000000; ++i) {
    total += PassByReference(s);
  }
  cout << total << endl;
  end_time = chrono::steady_clock::now();
  cout << chrono::duration<double>(end_time - start_time).count() << endl;
  return 0;
}
```

```
Output

260000000
2.6825
260000000
0.0452582
```

PassByValue:
A new string a is created and the content is copied from s

now duration

Cannot pass something other than variable by reference

```cpp
int PassByReference(string& s) {
  return s.length();
}
int main() {
  auto start_time = chrono::steady_clock::now();
  string s = "abcdefghijklmnopqrstuvwxyz";
  int total = 0;
  for (int i = 0; i < 10000000; ++i) {
    total += PassByReference(s);
    // total += PassByReference("123");
  }
  cout << total << endl;
  auto end_time = chrono::steady_clock::now();
  cout << chrono::duration<double>(end_time - start_time).count() << endl;
  return 0;
}
```

Compilation Error

```
Output

260000000
0.050572
```

Technical term:
To pass by reference, the argument must be an lvalue.

Pass by Const Reference

```cpp
int PassByConstReference(const string& s) {
  return s.length();
}
int main() {
  auto start_time = chrono::steady_clock::now();
  string s = "abcdefghijklmnopqrstuvwxyz";
  int total = 0;
  for (int i = 0; i < 10000000; ++i) {
    total += PassByConstReference(s);
    total += PassByConstReference("abc");
  }
  cout << total << endl;
  auto end_time = chrono::steady_clock::now();
  cout << chrono::duration<double>(end_time - start_time).count() << endl;
  return 0;
}
```

```
Output

290000000
1.36043
```

Don’t do this for primitives such as int, double, bool

struct

Use struct to declare data structures that can hold multiple data.

```
struct Point {
  double x, y;
};
int main() {
  Point p;
  p.x = 1.0;
  p.y = 2.5;
  Point q{3.6, 4.2};
  cout << q.x << " " << q.y << endl;
  return 0;
}
```

Braces initializer list

Follows the order of member variable declaration

```
Output
3.6 4.2
```

`struct` is a class with all members declared `public`

Don't forget the semi-colon ;

class

Constructor

If you declare your own constructor, default constructor will not be implicitly declared and defined.

```
struct Point {
  double x, y;
  Point(double d, double r) {
    x = d * cos(r);
    y = d * sin(r);
  }
};
int main() {
  Point p(2, 0.5235988);
  cout << p.x << " " << p.y << endl;
  // Point q;
  return 0;
}
```

Compilation Error

```
Output

1.73205 1
```


Constructors
Default constructor

Constructor

We can also use initializer list.

```
struct Point {
  double x, y;
  Point(double d, double r)
      : x(d * cos(r)), y(d * sin(r)) {}
};
int main() {
  Point p(2, 0.5235988);
  cout << p.x << " " << p.y << endl;
  return 0;
}
```



```
Output

1.73205 1
```

Member functions

```
struct Point {
  double x, y;
  Point(double d, double r)
      : x(d * cos(r)), y(d * sin(r)) {}
  double DistanceToOrigin() {
    return sqrt(x * x + y * y);
  }
  void Rotate() {
    swap(x, y);
    x = -x;
  }
};
int main() {
  Point p(2, 0.5235988);
  cout << p.x << " " << p.y << endl;
  cout << p.DistanceToOrigin() << endl;
  p.Rotate();
  cout << p.x << " " << p.y << endl;
  return 0;
}
```

```
Output

1.73205 1
2
-1 1.73205
```

swap

Overloading

```
struct Point {
  double x, y;
  Point Dot(double scalar) {
    return {scalar * x, scalar * y};
  }
  double Dot(const Point& p) {
    return x * p.x + y * p.y;
  }
};
int main() {
  Point p{2.2, 3.4};
  Point q = p.Dot(4.0);
  cout << q.x << " " << q.y << endl;
  Point r{1.5, 2.5};
  double s = p.Dot(r);
  cout << s << endl;
  return 0;
}
```

```
Output

8.8 13.6
11.8
```

Overload resolution

Operator overloading

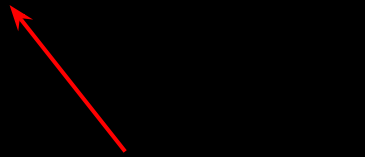
```
struct Point {
  double x, y;
  Point& operator*=(double scale) {
    x *= scale;
    y *= scale;
    return *this;
  }
};
int main() {
  Point p{2.2, 3.4};
  p *= 2.5;
  cout << p.x << " " << p.y << endl;
  return 0;
}
```

```
Output
5.5 8.5
```

Operator overloading

Operator overloading

```cpp
struct Point {
  double x, y;
  Point& operator*=(double scale) {
    x *= scale;
    y *= scale;
    return *this;
  }
};
ostream& operator<<(ostream& os, const Point& p) {
  os << p.x << " " << p.y;
  return os;
}
int main() {
  Point p{2.2, 3.4};
  p *= 2.5;
  cout << p << endl;
  return 0;
}
```




```
Output
5.5 8.5
```

operator<<

See section Stream extraction and insertion

Output a vector

```cpp
ostream& operator<<(ostream& os,
                    const vector<int>& c) {
  for (auto&& x : c) {
    cout << x << " ";
  }
  cout << endl;
  return os;
}
int main() {
  vector<int> a{1, 2, 3, 4, 5};
  vector<int> b{10, 11, 12};
  cout << a << b;
  return 0;
}
```

```
Output

1 2 3 4 5
10 11 12
```

Template


```cpp
template<class T>
ostream& operator<<(ostream& os,
                    const vector<T>& c) {
  for (auto&& x : c) {
    cout << x << " ";
  }
  cout << endl;
  return os;
}
int main() {
  vector<int> a{1, 2, 3, 4, 5};
  cout << a;
  vector<double> b{1.2, 3.4, 5.6};
  cout << b;
  return 0;
}
```

```
Output

1 2 3 4 5
1.2 3.4 5.6
```


Templates

Output a 2D vector

```cpp
template<class T>
ostream& operator<<(ostream& os,
                    const vector<T>& c) {
  for (auto&& x : c) {
    cout << x << " ";
  }
  cout << endl;
  return os;
}
int main() {
  vector<vector<int>> a{{11, 12, 13},
                        {21, 22},
                        {31, 32, 33}};
  cout << a;
  return 0;
}
```

```
Output

11 12 13
 21 22
 31 32 33
```

Pair

Pair can hold two values (`first`, `second`) of possibly different types.

Pairs can be compared. The first value will be compared first. If they are equal, the second value will be compared.

```cpp
int n;
cin >> n;
vector<pair<int, string>> students(n);
for (int i = 0; i < n; ++i) {
  cin >> students[i].first >> students[i].second;
}
sort(students.begin(), students.end());
for (auto& student : students) {
  cout << student.first << " ";
  cout << student.second << endl;
}
```

Input

```
4
3 Percy
2 Ian
3 Jeremy
1 Tony
```

Output

```
1 Tony
2 Ian
3 Jeremy
3 Percy
```

Pair

Pair can be useful to return multiple values.

```
pair<int, int> CountLetters(const string& s) {
  int upper = 0, lower = 0;
  for (char c : s) {
    upper += isupper(c) > 0;
    lower += islower(c) > 0;
  }
  return {upper, lower};
}
int main() {
  auto p = CountLetters("Hello, World!");
  cout << p.first << " " << p.second << endl;
  return 0;
}
```

```
Output

2 8
```


isupper islower

Tuple

What about more values?

```
tuple<int, int, int> CountLetters(const string& s) {
  int upper = 0, lower = 0, spaces = 0;
  for (char c : s) {
    upper += isupper(c) > 0;
    lower += islower(c) > 0;
    spaces += c == ' ';
  }
  return {upper, lower, spaces};
}
int main() {
  auto p = CountLetters("Hello, World!");
  cout << get<0>(p) << " " << get<1>(p) << " ";
  cout << get<2>(p) << endl;
  return 0;
}
```

```
Output

2 8 1
```

get(tuple)

J021 Date sorting

Given N dates, sort the dates in chronological order.

Input

```
3
4, July 1981
18, October 1982
22, December 1981
```

Output

```
4, July 1981
22, December 1981
18, October 1982
```

Reading the input

Let's read one line of input.

cin.get() reads the next character, which is comma here.

```
int day, year;
string month_string;
cin >> day;
cin.get();
cin >> month_string >> year;
cout << day << endl;
cout << month_string << endl;
cout << year << endl;
```

Input

```
20, February 2021
```

Output

```
20
February
2021
```

get

Converting the month into an integer

```cpp
const vector<string> kMonths =
    {"January", "February", "March", "April", "May", "June",
     "July", "August", "September", "October", "November", "December"};
int main() {
  int day, month, year;
  string month_string;
  cin >> day;
  cin.get();
  cin >> month_string >> year;
  auto it = find(kMonths.begin(), kMonths.end(), month_string);
  month = distance(kMonths.begin(), it);
  cout << day << endl;
  cout << month << endl;
  cout << year << endl;
  return 0;
}
```

Input

```
20, February 2021
```

Output

```
20
1
2021
```


find distance

Storing the dates in a vector<tuple<int, int, int>>

```cpp
const vector<string> kMonths =
    {"January", "February", "March", "April", "May", "June",
     "July", "August", "September", "October", "November", "December"};
int main() {
  int n;
  cin >> n;
  vector<tuple<int, int, int>> dates;
  for (int i = 0; i < n; ++i) {
    int day, month, year;
    string month_string;
    cin >> day;
    cin.get();
    cin >> month_string >> year;
    auto it = find(kMonths.begin(), kMonths.end(), month_string);
    month = distance(kMonths.begin(), it);
    dates.push_back({year, month, day});
    // dates.emplace_back(year, month, day);
  }
  cout << get<1>(dates[2]) << endl;
  return 0;
}
```

The most significant component should go first

You can also use `emplace_back`

```
Input

3
4, July 1981
18, October 1982
22, December 1981
```

```
Output

11
```

[emplace_back]

Alternative way

```cpp
const vector<string> kMonths =
    {"January", "February", "March", "April", "May", "June",
     "July", "August", "September", "October", "November", "December"};
int main() {
  int n;
  cin >> n;
  vector<tuple<int, int, int>> dates(n);
  for (auto& date : dates) {
    cin >> get<2>(date);
    cin.get();
    string month_string;
    cin >> month_string >> get<0>(date);
    auto it = find(kMonths.begin(), kMonths.end(), month_string);
    get<1>(date) = distance(kMonths.begin(), it);
  }
  cout << get<1>(dates[2]) << endl;
  return 0;
}
```

Input

```
3
4, July 1981
18, October 1982
22, December 1981
```

Output

```
11
```

Even fancier

```cpp
const vector<string> kMonths =
    {"January", "February", "March", "April", "May", "June",
     "July", "August", "September", "October", "November", "December"};
int main() {
  int n;
  cin >> n;
  vector<tuple<int, int, int>> dates(n);
  for (auto& [year, month, day] : dates) {
    cin >> day;
    cin.get();
    string month_string;
    cin >> month_string >> year;
    auto it = find(kMonths.begin(), kMonths.end(), month_string);
    month = distance(kMonths.begin(), it);
  }
  cout << get<1>(dates[2]) << endl;
  return 0;
}
```

Structured binding declaration
Must be auto

Input

```
3
4, July 1981
18, October 1982
22, December 1981
```

Output

```
11
```

Output the sorted dates

```
  vector<tuple<int, int, int>> dates(n);
  ... input ...
  sort(dates.begin(), dates.end());
  for (auto& date : dates) {
    cout << get<2>(date) << ", ";
    cout << kMonths[get<1>(date)] << " ";
    cout << get<0>(date) << endl;
  }
  return 0;
}
```

Input

```
3
4, July 1981
18, October 1982
22, December 1981
```

Output

```
4, July 1981
22, December 1981
18, October 1982
```

Solved with only 25 lines!

M2102 Social Distancing and miamia

Input	Output
4 &1.2 (120) {4} 1,,2,,3,,4,,8,,7,,6,,5,,	9.200000
5 &1 (16 0){8}1,2,3,4,5,6,7,8,1,8,2,7 ,{4}[1,8],[2,7],[3,6],(240)[4,5],	4.625000

M2102 Social Distancing and miamia

	Contestant	M2101 Social Distancing and Exam	M2102 Social Distancing and miamia
1		20 / 0:09	20 / 0:21
2		20 / 2:09	20 / 2:08
3	Why did I do q2 with Python	20 / 0:18	13
4		20 / 0:18	20 / 0:31
5		20 / 0:04	20 / 0:12
6		20 / 2:27	20 / 1:34
7		20 / 1:27	20 / 1:29
8		20 / 0:21	20 / 0:51
9		20 / 0:09	20 / 0:30
9		20 / 0:30	20 / 2:58

Reading the start time

```cpp
int main() {
  int n;
  cin >> n;
  char c;
  cin >> c;
  double current_time;
  cin >> current_time;
  cout << current_time<< endl;
  return 0;
}
```

Input

```
4
&1.2
(120)
{4}
1,,2,,3,,4,,8,,7,,6,,5,,
```

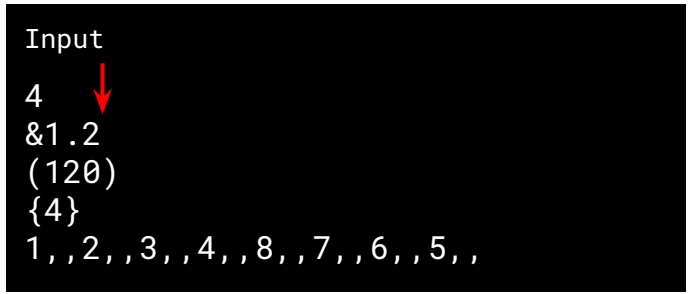
Output

```
1.2
```

Reading the rest of the data - Method 1

```
...
  string s;
  for (int i = 1; i < n; ++i) {
    string t;
    cin >> t;
    s += t;
  }
  cout << s << endl;
```

Input



Output

```
(120){4}1,,2,,3,,4,,8,,7,,6,,5,,
```

Reading the rest of the data - Method 2

```
int main() {
  int n;
  cin >> n;
  char c;
  cin >> c;
  double current_time;
  cin >> current_time;
  string s = accumulate(istream_iterator<string>(cin),
                istream_iterator<string>(), string());
  cout << s << endl;
  return 0;
}
```

End of stream

Empty string

```
Input
4
&1.2
(120)
{4}
1,,2,,3,,4,,8,,7,,6,,5,,
```

```
Output
(120){4}1,,2,,3,,4,,8,,7,,6,,5,,
```

Uses operator + to concatenate strings.

Note: press CTRL+Z (windows) / CTRL+D (linux) for end-of-file

accumulate
istream_iterator

Tokenize the input

Break down the string into space separated tokens.

```
(120){4}1,,2,,3,,4,,8,,7,,6,,5,,
```

```
bpm 120 notevalue 4 1,,2,,3,,4,,8,,7,,6,,5
```

```
(160){8}1,2,3,4,5,6,7,8,1,8,2,7,{4}[1,8],[2,7],[3,6],(240)[4,5],
```

```
bpm 160 notevalue 8 1,2,3,4,5,6,7,8,1,8,2,7, notevalue 4 [1,8] ,
[2,7] , [3,6] , bpm 240 [4,5] ,
```

Tokenize the input

```
stringstream ss;
for (char c : s) {
  if (c == '(') {
    ss << " bpm ";
  } else if (c == '{') {
    ss << " note_value ";
  } else if (c == '[') {
    ss << " [";
  } else if (c == ')' || c == '}' || c == ']') {
    ss << " ";
  } else {
    ss << c;
  }
}
cout << ss.str() << endl;
```

Input

```
5
&1
(16
0){8}1,2,3,4,5,6,7,8,1,8,2,7
,{4}[1,8],[2,7],[3,6],(240)[4,5
],
```

Output

```
 bpm 160  notevalue 8
1,2,3,4,5,6,7,8,1,8,2,7,
notevalue 4  [1,8 , [2,7 , [3,6 ,
bpm 240  [4,5 ,
```

Change closing brackets to whitespace

stringstream

Process BPM and note value

```
double bpm = 0, note_value = 0;
while (!ss.eof()) {
  string token;
  ss >> token;
  if (token == "bpm") {
    ss >> bpm;
    cout << bpm << endl;
  } else if (token == "note_value") {
    ss >> note_value;
    cout << note_value << endl;
  }
}
```

`stringstream` is useful for type conversions

Input

```
5
&1
(16
0){8}1,2,3,4,5,6,7,8,1,8,2,7
,{4}[1,8],[2,7],[3,6],(240)[4,5
],
```

ss

```
 bpm 160  notevalue 8
1,2,3,4,5,6,7,8,1,8,2,7,
notevalue 4  [1,8 , [2,7 , [3,6 ,
bpm 240  [4,5 ,
```

Output

```
160
8
4
240
```

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Process beats and ignore brackets

```
double bpm = 0, note_value = 0;
while (!ss.eof()) {
  string token;
  ss >> token;
  if (token == "bpm") {
    ss >> bpm;
  } else if (token == "note_value") {
    ss >> note_value;
  } else if (token[0] != '[') {
    int commas = count(token.begin(), token.end(), ',');
    current_time += commas * 240.0 / bpm / note_value;
  }
}
cout << fixed << setprecision(9) << current_time << endl;
```

Set to fixed point format (default = scientific notation)

Precision = 9 d.p. is sufficient.

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Input

```
5
&1
(16
0){8}1,2,3,4,5,6,7,8,1,8,2,7
,{4}[1,8],[2,7],[3,6],(240)[4,5
],
```

ss

```
 bpm 160  notevalue 8
1,2,3,4,5,6,7,8,1,8,2,7,
notevalue 4  [1,8 , [2,7 , [3,6 ,
bpm 240  [4,5 ,
```

Output

```
4.625000000
```

count fixed setprecision

Solved?

Sadly, the 40 line program fails to solve the task :(

because `operator+` creates a new string every time.

This will be “fixed” in C++20 where the intermediate value will be moved using `std::move`.

[accumulate]

C++17

Test	Result	Run Time
1	Accepted	0.001 s
2	Accepted	0.708 s
3	Time Limit Exceeded	1.000 s
4	Time Limit Exceeded	1.000 s
1	Accepted	0.002 s
2	Accepted	0.151 s
3	Time Limit Exceeded	1.000 s
4	Time Limit Exceeded	1.000 s
5	Time Limit Exceeded	1.000 s
1	Accepted	0.001 s
2	Accepted	0.031 s
3	Accepted	0.883 s
4	Time Limit Exceeded	1.000 s
5	Time Limit Exceeded	1.000 s
1	Accepted	0.002 s
2	Accepted	0.040 s
3	Time Limit Exceeded	1.000 s
4	Time Limit Exceeded	1.000 s
5	Time Limit Exceeded	1.000 s
6	Accepted	0.246 s

C++20

Test	Result	Run Time
1	Accepted	0.001 s
2	Accepted	0.018 s
3	Accepted	0.052 s
4	Accepted	0.216 s
1	Accepted	0.001 s
2	Accepted	0.008 s
3	Accepted	0.041 s
4	Accepted	0.096 s
5	Accepted	0.399 s
1	Accepted	0.001 s
2	Accepted	0.005 s
3	Accepted	0.019 s
4	Accepted	0.156 s
5	Accepted	0.263 s
1	Accepted	0.001 s
2	Accepted	0.005 s
3	Accepted	0.052 s
4	Accepted	0.191 s
5	Accepted	0.625 s
6	Accepted	0.245 s

Closing

Use a lot of library functions != Good programs

C++ style guide

Google C++ style guide

Indentation: 2 spaces

1 space around binary operators

1 space before and after parenthesis ()

etc.

Macros

Some competitive programmers use macros to shorten their code.

```
#define x first
#define y second
#define pii pair<int,int>
#define ll long long
#define pll pair<ll,ll>
#define pbb pair<bool,bool>
#define mp make_pair
#define pb push_back
#define pf push_front
#define popb pop_back
#define popf pop_front
#define xmod (ll)(1e9+7)
#define hmod 1286031825167LL
```

This is discouraged for several reasons:

- It makes code hard to read for others
- It makes the code longer (harder to find main)
- It is easy to introduce subtle bugs
- It makes debugging harder

dxxxxxe

```
#include<bits/stdc++.h>
using namespace std;
typedef array<int,4> arin;
vector<arin>v;
bool sw=false;
void out(int a,int b,int c,int d){
    if(!sw) v.push_back({a,b,c,d});
    else v.push_back({b,a,d,c});
```

mxxxxxg

```
#include<bits/stdc++.h>
using namespace std;
int r,c;
vector< pair< pair<int,int>, pair<int,int
bool a[105][105];
int dx[8] = {0, 0, 1, -1, 1, 1, -1, -1};
int dy[8] = {1, -1, 0, 0, -1, 1, -1, 1};
```

```
/**
 *    author:  tourist
 *    created: 28.01.2021 19:09:28
**/
#include <bits/stdc++.h>

using namespace std;

int main() {
```

By Benq, contest: Educational Cod

```
#include <bits/stdc++.h>
using namespace std;

// returns the first index
int firstAtLeast(const vect
```

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Exercises

01007 Packet Re-assembly

01009 Words

M1902 Zero and Scheduling Problem

M2001 Corona and WFH