C++ fundamentals for competitive programming

CPPoliTO

- Ghassane Ben El Aattar
- Francesco Imparato
- Nima Naderi Ghotbodini

General C++ code in CP

```
#include <bits/stdc++.h>
using namespace std;
int main{
    int a, b;
    cin >> a >> b;
    int ans = a + b;

    cout << a + b << '\n';
}
```

General C++ code in CP

```
#include <bits/stdc++.h>
using namespace std;
int main{
    int a, b;
    cin >> a >> b;
    int ans = a + b;

    cout << a + b << '\n';
}
```

semicolon
“;”

Input/Output (IO) in C++

cin & cout

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

int main() {
    int num;
    cout << "Enter a number: ";
    // Unlike Python Print, cout wouldn't add '\n' (endl) at the end
    cin >> num;
    cout << "You entered: " << num << endl;

    // you can get two variable in the input
    int a, b;
    cin >> a >> b;
  // can be given in the same line or two separate lines!


    return 0;
}
```

Other useful things you can learn

Other useful things you can learn

- printf("%d\n", 83); scanf("%d", &age);

Other useful things you can learn

- printf("%d\n", 83);
- getchar();

scanf("%d", &age);
putchar(ch);

Other useful things you can learn

- printf("%d\n", 83);
- getchar();
- getline(cin, line_1);

scanf("%d", &age);

putchar(ch);

Other useful things you can learn

- printf("%d\n", 83); scanf("%d", &age);
- getchar(); putchar(ch);
- getline(cin, line_1);
- fast_io
  - ios::sync_with_stdio(0);
  - cin.tie(0);
  - cout.tie(0);

Other useful things you can learn

- printf("%d\n", 83); scanf("%d", &age);
- getchar(); putchar(ch);
- getline(cin, line_1);
- fast_io
  - ios::sync_with_stdio(0);
  - cin.tie(0);
  - cout.tie(0);
- file_io
  - freopen("input.txt" , "r+" , stdin);
  - freopen("output.txt" , "w+" , stdout);

Variables in C++

Variables in C++

In C++, there are several different types of variables that can be used to store different kinds of data. Here are some of the most common variable types in C++:

Variables in C++

In C++, there are several different types of variables that can be used to store different kinds of data. Here are some of the most common variable types in C++:

1. int
2. long long
3. double
4. char
5. bool
6. strings

Variables in C++

In C++, there are several different types of variables that can be used to store different kinds of data. Here are some of the most common variable types in C++:

1. int
2. long long
3. double
4. char
5. bool
6. strings

C++ also has short int, long int, signed and unsigned int/char you can learn more about them in this link: https://www.programiz.com/cpp-programming/type-modifiers

int

Used to store integer values, such as whole numbers. For example, "int x = 10;" declares a variable called x that stores the value 10.

int

Used to store integer values, such as whole numbers. For example, "int x = 10;" declares a variable called x that stores the value 10.

An "int" data type typically uses 4 bytes (32 bit) of memory and can represent integers in the range of -2,147,483,648 to 2,147,483,647.

int

Used to store integer values, such as whole numbers. For example, "int x = 10;" declares a variable called x that stores the value 10.

An "int" data type typically uses 4 bytes (32 bit) of memory and can represent integers in the range of -2,147,483,648 to 2,147,483,647.

-2e9 to 2e9

int

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

int main() {
    int x = 5;
    int y = 7;
    int z = x + y;

    cout << "The value of x is " << x << endl;
    cout << "The value of y is " << y << endl;
    cout << "The value of z is " << z << endl;

    return 0;
}
```

long long

Used to store very big integer values. For example,

"long long x = 1e18;" declares a variable called x that stores the value 10^18.

long long

Used to store very big integer values. For example,

"long long x = 1e18;" declares a variable called x that stores the value 10^18.

An "long long" data type typically uses 8 bytes (64 bit) of memory and can represent integers in the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

long long

Used to store very big integer values. For example,

"long long x = 1e18;" declares a variable called x that stores the value 10^18.

An "long long" data type typically uses 8 bytes (64 bit) of memory and can represent integers in the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

-8e18 to 8e18

long long

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

int main() {
    long long x = 1e9 + 7;
    long long y = 1e9 + 9;
    long long z = x * y; // no overflow

    cout << "The value of x is " << x << endl;
    cout << "The value of y is " << y << endl;
    cout << "The value of z is " << z << endl;

    /*
    tip :
        '1e9' is double value but when we
        write "x = 1e9", '1e9' would be casted
        to long long value type
    */
    return 0;
}
```

double

Used to store floating-point numbers, which are numbers with a decimal point. For example, "double pi = 3.14;" declares a variable called pi that stores the value 3.14

double

Used to store floating-point numbers, which are numbers with a decimal point. For example, "double pi = 3.14;" declares a variable called pi that stores the value 3.14

A "double" typically uses 8 bytes of memory.

In general, a "double" can store values in the range of approximately 10^-308 to 10^308, with a precision of about 15-16 digits.

double & long double

precision of a "double" can be affected by factors such as the specific values being stored, the operations being performed on those values, and any potential rounding errors!

double & long double

precision of a "double" can be affected by factors such as the specific values being stored, the operations being performed on those values, and any potential rounding errors!

A "long double", on the other hand, typically uses 10 or 12 bytes of memory (depending on the implementation) and can store values in the range of approximately 10^-4932 to 10^4932, with a precision of about 19-20 digits.
This provides much greater precision and range than a "double".

double & long double

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

int main() {
    double       d = 3.14159265358979323846;
  // Declare and initialize a double precision floating-point variable
    long double ld = 3.14159265358979323846;
  // Declare and initialize a long double precision floating-point variable

    cout << setprecision(20) << fixed;
    // Set output format to 20 decimal places and fixed point notation

    cout << "The value of d is " << d << endl; // Output the value of "d" to the console
    cout << "The value of ld is " << ld << endl; // Output the value of "ld" to the console

    return 0;
}
```

double & long double

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

int main() {
    double       d = 3.14159265358979323846;
  // Declare and initialize a double precision floating-point variable
    long double ld = 3.14159265358979323846;
  // Declare and initialize a long double precision floating-point variable

    cout << setprecision(20) << f
    // Set output format to 20 de

    cout << "The value of d is "
    cout << "The value of ld is "

    return 0;
}
```

```
The value of d is 3.14159265358979311600
The value of ld is 3.14159265358979323846
```

Castings and type conversions

Be careful when writing operations between different number types!

- If you need a double variable you have first to cast to double the integer variables
  - int a = 2; double b = (double)a / 3.0;
  - Why “/ 3.0” and not “/ 3”? Because otherwise it will be seen as an int

char

Used to store single characters, such as letters or symbols. For example, "char letter = 'A';" declares a variable called letter that stores the value 'A'.

char

Used to store single characters, such as letters or symbols. For example, "char letter = 'A';" declares a variable called letter that stores the value 'A'.

A "char" variable uses 1 byte (8 bit) of memory and can store 256 different values, ranging from 0 to 255

char

In addition to representing individual characters, "char" can also be used to represent integer values using the ASCII encoding.
For example, the character 'a' has an ASCII value of 97.

char

In addition to representing individual characters, "char" can also be used to represent integer values using the ASCII encoding.
For example, the character 'a' has an ASCII value of 97.

```
cook@pop-os:~$ ascii -d
    0 NUL    16 DLE    32       48 0    64 @    80 P    96 `   112 p
    1 SOH    17 DC1    33 !     49 1    65 A    81 Q    97 a   113 q
    2 STX    18 DC2    34 "     50 2    66 B    82 R    98 b   114 r
    3 ETX    19 DC3    35 #     51 3    67 C    83 S    99 c   115 s
    4 EOT    20 DC4    36 $     52 4    68 D    84 T   100 d   116 t
    5 ENQ    21 NAK    37 %     53 5    69 E    85 U   101 e   117 u
    6 ACK    22 SYN    38 &     54 6    70 F    86 V   102 f   118 v
    7 BEL    23 ETB    39 '     55 7    71 G    87 W   103 g   119 w
    8 BS     24 CAN    40 (     56 8    72 H    88 X   104 h   120 x
    9 HT     25 EM     41 )     57 9    73 I    89 Y   105 i   121 y
   10 LF     26 SUB    42 *     58 :    74 J    90 Z   106 j   122 z
   11 VT     27 ESC    43 +     59 ;    75 K    91 [   107 k   123 {
   12 FF     28 FS     44 ,     60 <    76 L    92 \   108 l   124 |
   13 CR     29 GS     45 -     61 =    77 M    93 ]   109 m   125 }
   14 SO     30 RS     46 .     62 >    78 N    94 ^   110 n   126 ~
   15 SI     31 US     47 /     63 ?    79 O    95 _   111 o   127 DEL
```

char & int ASCII

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

int main() {
    char c = 'A';
  // Declare and initialize a char variable with the value 'A'

    cout << "The value of c is " << c << endl;
  // Output the value of "c" to the console

    int ascii_value = c;
  // Store the ASCII value of "c" in an integer variable
    cout << "The ASCII value of c is " << ascii_value << endl;
  // Output the ASCII value of "c" to the console

    return 0;
}
```

char & int ASCII

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

int main() {
    char c = 'A';
 initialize a char variable with the value 'A'

 value of c is " << c << endl;
 value of "c" to the console

lue = c;
SCII value of "c" in an integer variable
 ASCII value of c is " << ascii_value << endl;
ASCII value of "c" to the console
```

```
The value of c is A
The ASCII value of c is 65
```

bool

Used to store boolean values, which are either true(1) or false(0).

For example, "bool isTrue = true;" declares a variable called isTrue that stores the value true.

bool

Used to store boolean values, which are either true(1) or false(0).

For example, "bool isTrue = true;" declares a variable called isTrue that stores the value true.

A "bool" variable uses 1 byte (8 bits, naively 8 bits not only 1!) of memory and can store only two possible values: "true" or "false".

In C++, "false" is represented by the value 0 and "true" is represented by any non-zero value.

bool

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

int main() {
    bool is_greater_than = (5 > 3);
  // Declare and initialize a bool variable with the result of a comparison

    cout << "is 5 greater than 3? " << is_greater_than << endl;

    return 0;
}
```

bool

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

int main() {
    bool is_greater_than = (5 > 3);
  // Declare and initialize a bool variable with the result of a comparison

    cout << "is 5 greater than 3? " << is_greater_than << endl;

    return 0;
}
```

```
is 5 greater than 3? 1
```

strings

Used to store a sequence of characters, such as a word or sentence. For example, "string name = "John";" declares a variable called name that stores the value "John".

strings

Used to store a sequence of characters, such as a word or sentence. For example, "string name = "John";" declares a variable called name that stores the value "John".

Unlike python, strings in C++ are mutable, which means you can change i’th character of the string!

strings

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

int main() {
    string word = "hello";

    cout << "Original word: " << word << endl;

    word[0] = 'j';
    word[3] = 'p';

    cout << "Changed word: " << word << endl;

    return 0;
}
```

strings

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

int main() {
    string word = "hello";

    cout << "Original word:

    word[0] = 'j';
    word[3] = 'p';

    cout << "Changed word: "

    return 0;
}
```

```
Original word: hello
Changed word: jelpo
```

Pair of two variables

In C++, a pair is a container that holds two values.

Pair of two variables

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

int main() {
    // Declaring and initializing a pair
    pair<string, int> myPair("Alice", 25);
    cout << "Name: " << myPair.first << endl;
    cout << "Age: " << myPair.second << endl;

    // Modifying a pair
    myPair.first = "Bob";
    myPair.second = 30;
    cout << "Name: " << myPair.first << endl;
    cout << "Age: " << myPair.second << endl;

    return 0;
}
```

Basic Statements in C++

if & else

"if-else" statement: The "if-else" statement is a control flow statement that allows you to execute different blocks of code based on a specified condition. The basic syntax of an "if-else" statement in C++ is:

"if-else
statem
based c
statem

```
#include <iostream>

using namespace std;

int main() {
    int age;

    cout << "Enter your age: ";
    cin >> age;

    if (age < 0) {
        cout << "Invalid age. Please enter a positive value." << endl;
    } else if (age < 18) {
        cout << "You are a minor." << endl;
    } else if (age < 65) {
        cout << "You are an adult." << endl;
    } else {
        cout << "You are a senior citizen." << endl;
    }

    return 0;
}
```

y
de
se"

while loop

"while" loop: The "while" loop is another control flow statement that allows you to execute a block of code repeatedly based on a specified condition. It is typically used when you don't know the number of times you want to execute the loop. The basic syntax of a "while" loop in C++ is:

"while" loop: Th l flow statement that allows you atedly based on a specified cond ou don't know the number of ti op. The basic syntax of a "whi

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

int main() {
    int num = 1;

    while (num <= 10) {
        cout << num << " ";
        num++;
    }

    return 0;
}
```

for loop

"for" loop: The "for" loop is a control flow statement that allows you to execute a block of code repeatedly based on a specified condition. It is typically used when you know the number of times you want to execute the loop. The basic syntax of a "for" loop in C++ is:

for loop

"for" loop: The "for" loop is a control flow statement that allows you to execute a block of code repeatedly based on a specified condition. It is typically used when you know the number of times you want to execute the loop. The basic syntax of a "for" loop in C++ is:

```
for (initialization; condition; increment) {
    // Code to be executed
}
```

for loop

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

int main() {

    for (int i = 1; i <= 10; i ++) {
        cout << i << " ";
    }

    for (int i = 10; i >= 1; i --) {
        cout << i << " ";
    }

    return 0;
}
```

continue & break in loops

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

int main() {

    for (int i = 1; i <= 10; i ++) {
        if(i == 5) continue;
        cout << i << " ";
    }

    for (int i = 10; i >= 1; i --) {
        if(i == 3) break;
        cout << i << " ";
    }

    return 0;
}
```

C++ Standard Template Library

Standard template library (STL)

C++ offers a library of various algorithm and data structures which can be easily used. We will go over the basic elements of the main STL algorithms and data structures.

Standard template library (STL)

C++ offers a library of various algorithm and data structures which can be easily used. We will go over the basic elements of the main STL algorithms and data structures.

- Dynamic arrays (vectors);

Standard template library (STL)

C++ offers a library of various algorithm and data structures which can be easily used. We will go over the basic elements of the main STL algorithms and data structures.

- Dynamic arrays (vectors);
- Sorting;

Standard template library (STL)

C++ offers a library of various algorithm and data structures which can be easily used. We will go over the basic elements of the main STL algorithms and data structures.

- Dynamic arrays (vectors);
- Sorting;
- Queue, stack, set, map, etc..

Vectors (1)

C++ vectors are dynamic arrays which are resized automatically. They can be defined in the following ways:

Vectors (1)

C++ vectors are dynamic arrays which are resized automatically. They can be defined in the following ways:

- To create an empty vector of integers, use: **vector<int> v;**

Vectors (1)

C++ vectors are dynamic arrays which are resized automatically. They can be defined in the following ways:

- To create an empty vector of integers, use: **vector<int> v;**
- To create a vector of integers of fixed number of elements (20 for example), use: **vector<int> v(20);**

Vectors (1)

C++ vectors are dynamic arrays which are resized automatically. They can be defined in the following ways:

- To create an empty vector of integers, use: **vector<int> v;**
- To create a vector of integers of fixed number of elements (20 for example), use: **vector<int> v(20);**
- To create a vector of integers with specific elements, use: **vector<int> v = {4,2,5,3,5,8,3};**

Vectors (2)

Vectors are 0-indexed and it’s possible to access each element by its index, like regular C style arrays.

Vectors (2)

Vectors are 0-indexed and it's possible to access each element by its index, like regular C style arrays.

One of the main operations on a vector is appending an element at the end of it, by using the **push_back** function.

Vectors (2)

Vectors are 0-indexed and it's possible to access each element by its index, like regular C style arrays.

One of the main operations on a vector is appending an element at the end of it, by using the **push_back** function.

For example, we can append 20 to a vector of integers **v** by doing: **v.push_back(20);**

Sorting

C++ like many other languages has a built-in STL function for sorting an array.

Sorting

C++ like many other languages has a built-in STL function for sorting an array.

- To sort a vector, we can use **sort(v.begin(),v.end()))**; to sort in increasing order, or **sort(v.rbegin(),v.rend()); to sort in reverse order.

Sorting

C++ like many other languages has a built-in STL function for sorting an array.

- To sort a vector, we can use **sort(v.begin(),v.end()**); to sort in increasing order, or **sort(v.rbegin(),v.rend());** to sort in reverse order.
- To sort a C style array we can use **sort(a,a+n);** where n is the size of the array.

Queue

To define a queue of integers, we can define queue<int> q. The main functions available are:

Queue

To define a queue of integers, we can define queue<int> q. The main functions available are:

- **q.push(x)** to push an integer x at the end of the queue;

Queue

To define a queue of integers, we can define queue<int> q. The main functions available are:

- **q.push(x)** to push an integer x at the end of the queue;
- **q.top()** to access the oldest element inserted in the queue;

Queue

To define a queue of integers, we can define queue<int> q. The main functions available are:

- **q.push(x)** to push an integer x at the end of the queue;
- **q.top()** to access the oldest element inserted in the queue;
- **q.pop()** to remove from the queue the oldest element inserted.

Map

We can define a Map with integer keys and values with: **map<int, int> m**.

Map

We can define a Map with integer keys and values with: **map<int, int> m**. This is the binary search tree (RBT) implementation with logarithmic access.

Map

We can define a Map with integer keys and values with: **map<int, int> m**. This is the binary search tree (RBT) implementation with logarithmic access. We can define the hashmap with **unordered_map<int, int>**, but it's not recommended for CP.

Map

We can define a Map with integer keys and values with: **map<int, int> m**. This is the binary search tree (RBT) implementation with logarithmic access. We can define the hashmap with **unordered_map<int, int>**, but it's not recommended for CP.

Some of the operations possible for the STL map are:

Map

We can define a Map with integer keys and values with: **map<int, int> m**. This is the binary search tree (RBT) implementation with logarithmic access. We can define the hashmap with **unordered_map<int, int>**, but it's not recommended for CP.

Some of the operations possible for the STL map are:

- insert an element by key using **m[x] = y**;

Map

We can define a Map with integer keys and values with: **map<int, int> m**. This is the binary search tree (RBT) implementation with logarithmic access. We can define the hashmap with **unordered_map<int, int>**, but it's not recommended for CP.

Some of the operations possible for the STL map are:

- insert an element by key using **m[x] = y**;
- modify an element by key using **m[x] = z**;

Map

We can define a Map with integer keys and values with: **map<int, int> m**. This is the binary search tree (RBT) implementation with logarithmic access. We can define the hashmap with **unordered_map<int, int>**, but it's not recommended for CP.

Some of the operations possible for the STL map are:

- insert an element by key using **m[x] = y**;
- modify an element by key using **m[x] = z**;
- erase a key with **m.erase(x)**.

Set

C++ set is a container which stores unique elements in increasing order and it’s define as **set<int> s** for integer sets. It’s possible to insert and erase element, and also accessing the smallest and biggest elements from a set.

Set

C++ set is a container which stores unique elements in increasing order and it's define as **set<int> s** for integer sets. It's possible to insert and erase element, and also accessing the smallest and biggest elements from a set.
Let's look at some functions for integer sets:

Set

C++ set is a container which stores unique elements in increasing order and it's define as **set<int> s** for integer sets. It's possible to insert and erase element, and also accessing the smallest and biggest elements from a set.
Let's look at some functions for integer sets:

- Insert or delete an element x with **s.insert(x)** and **s.erase(x)**

Set

C++ set is a container which stores unique elements in increasing order and it's define as **set<int> s** for integer sets. It's possible to insert and erase element, and also accessing the smallest and biggest elements from a set.
Let's look at some functions for integer sets:

- Insert or delete an element x with **s.insert(x)** and **s.erase(x)**
- Get the iterator pointing to the first or last elements by using **s.begin()** and **s.rbegin()**.

Iterations

It’s possible to iterate through elements of an iterable container, such as vectors, sets, maps in the following way:

```
for(auto x : cont){

}
```

Templates in C++

You can create your own Template based on your preferences

```
#pragma GCC optimize("O2")
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef pair<ll, ll> pll;
const ll MXN = 400 + 10;

int main(){
    ios::sync_with_stdio(0);cin.tie(0); cout.tie(0);


    return 0;
}
//! N.N
```

https://codeforces.com/blog/entry/77199

Thank you for your attention!

CPPoliTO