C++ language reference

Note: A work in progress. Visit https://pbaumgarten.com/cpp for the latest version.

Basic C++ syntax

```
#include <iostream>    // include system library
using namespace std;

int main() {
   int num = 0;
   cout << "Hello world\n";
   return 0;
}
```

Compile & execute

To compile

```
g++ -o hello.exe hello.cpp
```

To compile, linking a library that is in the standard path

```
g++ -o hello.exe hello.cpp -lmylib
```

To compile, linking a library that is not in the standard path (folder containing *mylib.so*)

- `-I` folder containing custom `.h` files
- `-L` folder containing the associated object files eg: `.so .dll .a`
- `-l` actual library to import (the `xxxx` portion of the `libxxxx.so` filename)

```
g++ -o hello.exe hello.cpp -I/my/project/include -L/my/project/lib -lmylib
```

To compile and execute on Mac/Linux environment

```
g++ -o hello hello.cpp
./hello
```

Just out of pure interest... if you want to see what your C/C++ code compiles to at an assembler level, this is a fun website to try: https://godbolt.org/z/4G94Es

For HKOI participants

For all questions you can assume the following initialising code:

```
#include <cstdio>
#include <cmath>
#include <cstdlib>
#include <string>
#include <iostream>
using namespace std;
```

Primitive data types

Common term	C type	Num of bits	Max unsigned
Boolean	`bool`	32	Can only be **true** (1) or **false** (0). Internally stored as an integer
Byte	`char`	8	255
Word	`short`	16	65 535
Integer	`int`	32	4 294 967 295
Long Integer	`long`	32 / 64	4 294 967 295 / ~18 quintillion
Long Long Int	`long long`	64	~18 quintillion
Float	`float`	32	3.402823466 e+38
Double Float	`double`	64	1.7976931348623158 e+308
Pointer	`* x`	32 / 64	~4 billion / ~18 quintillion
String	`string`	varies	c++ class (*not a primitive, but so common it's worth listing here*)

Declaring a normal primitive

```
int bob = 42;
```

Constants should be prefixed with keyword **const**

```
const int maxPeople = 5;
```

Casting

```
string s = to_string(42);        // Integer (or other numeric type) to c++ class string
int i = stoi("42");              // String to integer
long l = stol("42");             // String to long
float f = stof("3.1415");        // String to float
double d = stod("3.1415");       // String to double
```

Operators

`+ - * / %`	Arithmetic operations
`++i --i`	Pre-increment or pre-decrement of i
`i++ i--`	Increment or decrement of i
`&& ||`	Logical operations: and, or
`& | ^ ! ~`	Bit logical operations: and, or, xor, not, complement (swap 1s & 0s)
`>> <<`	Bit shift right and left: `int n=16; n = n << 2;` computes 64.
`=`	Assignment operator
`+= -= *= etc`	Shorthand operation. `i += 2;` is equiv to `i = i + 2;`
`== != < > <= >=`	Comparison operators (useful only on primitive types)
`(,)`	Computed value is last. Eg: `a = (b,c,d);` Executes b,c,d then assigns a=d

Control structures

If / else

```
int a,b;

if (a > b) {
   cout << "a is greater\n";
} else if (a < b) {
   cout << "b is greater\n";
} else {
   cout << "a and b are equal\n";
}
```

While

```
int i=0;
while (i < 10) {
   cout << i << "\n";
   i++;
}
```

For

```
for (int i=0; i<10; i++) {
   cout << i << "\n";
}
```

Do while

```
int i=0;
do {
   cout << i << "\n";
} while (i < 10);
```

Switch case

```
switch (i) {
   case 0:  cout << "is 0\n"; break;
   case 1:  cout << "is 1\n"; break;
   case 2:  cout << "is 2\n"; break;
   default: cout << "none of the above\n"; break;
}
```

Ternary (aka the "in-line if") OFTEN NOT PERMITTED IN HKOI

```
// if (a > b) then result = x else result = y
result = a > b ? x : y;
```

I/O streams

Standard input is recommended only for simple/testing programs as there is no real capacity for error checking.

```
#include <iostream>
int main() {
   int num;
   std::cout << "Type a number: ";
   std::cin >> num;
   std::cout << "You typed" << num << "\n";
}
```

Note:

```
cin >> a >> b;
```

Is equivalent to

```
cin >> a;
cin >> b;
```

Issues with cin:

- Any kind of space is used to separate two consecutive input operations; this may either be a space, a tab, or a new-line character.
- cin extraction always considers spaces (whitespaces, tabs, new-line...) as terminating the value being extracted, and thus extracting a string means to always extract a single word, not a phrase or an entire sentence.

To input a full line without breaking on spaces, use the **getline** function....

```
string s;
cout << "What's your name? ";
getline(cin, s);
cout << "Hello " << s << "\n";
```

When performance matters, some recommendations from https://stackoverflow.com/a/18688906

- Turn off sync with stdio.
  `std::ios_base::sync_with_stdio(false);`
- Use `"\n"` instead of `std::endl`

Math

```
#include <cmath>
```

- Link using `-lm` when compiling
- All functions take and return double unless otherwise noted

```
// Rounding etc
abs(x)      // absolute value
ceil(x)     // smallest integer (returned as double) no less than x
floor(x)    // largest integer (returned as double) no greater than y
trunc(f)    // truncate decimals off float into an integer
round(f)    // round a float into an integer
```

```
// Exponents and logs
log(x)      // natural logarithm of x
log2(x)     // log base 2
log10(x)    // log base 10
exp(x)      // e to the power of x
exp2(x)     // 2 to the power of x
exp10(x)    // 10 to the power of x
pow(x,y)    // x to the power of y
sqrt(x)     // square root of x
```

```
// Trigonometry
sin(a)      // sine in radians
cos(a)      // cosine in radians
tan(a)      // tangent of double (in radians)
asin(x)    // arcsine of x
acos(x)     // arccosine of x
atan(x)     // arctangent of x
atan2(y,x)  // principal inverse of tan(y/x) in same quadrant as (x,y)
hypot(x,y)  // hypotenuse of triangle with sides x,y
```

```
// Easy pseudo-random numbers
rand()                      // pseudo-random integer between 0 and RAND_MAX (usually 32767)
(rand()/(RAND_MAX + 1.0))   // pseudo-random float between 0 and 1
rand() % 100                // pseudo-random integer between 0 and 99
                            // Warning: the distribution will slightly favour numbers < 67
                            // 1 in 327 times with this approach (may or may not matter)
```

```
// Safer random numbers     // Read https://stackoverflow.com/a/19666713
#include <random>
std::random_device rd;
std::mt19937 mt(rd());
std::uniform_real_distribution<double> dist(1.0, 10.0);
double randomNumber = dist(mt); // between 1.0 and 10.0
```

```
#include <utility>
swap(a,b);                  // Swap two values using reference passing
```

Strings

C++ provides following two types of string representations –

- The C-style character string.
- The String class type introduced with Standard C++.

The C-Style Character String (also known as null terminated strings)

```
#include <cstring>
char greeting[] = "Hello";
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; // equivalent statements
```

Functions for C strings

```
strcpy(s1, s2);     // Copies string s2 into string s1.
strcat(s1, s2);     // Concatenates string s2 onto the end of string s1.
strlen(s1);         // Returns the length of string s1.
strcmp(s1, s2);     // Returns 0 if s1 and s2 are the same, less than 0 if s1<s2,
                    // greater than 0 if s1>s2.
strchr(s1, ch);     // Returns a pointer to first occurrence of character ch in string s1.
strstr(s1, s2);     // Returns a pointer to first occurrence of string s2 in string s1.
```

String class strings

```
#include <string>

string s1 = "abcde";
string s2 = "vwxyz";
string s3;

s3 = s1;                                // copy s1 into s3
s3 = s1 + s2;                           // concatenates s1 and s2 into s3
int len = s3.size();                    // total length of s3
int len = s3.length();                  // total length of s3... yes, same as size()
char c = s1.at(3);                      // get character at position 3. 'd'
s1.append(s2, 3);                       // append 3 chars of s2 to end of s1. 'abcdevwx'
s1.insert(3, s2);                       // insert s2 into s1 from pos 3. 'abcvwxyzde'
s1.replace(3, 2, s2);                   // Replace portion that begins at char 9 and spans 5
                                        // characters. 'abcvwxyz'
int pos = s1.find_first_of("cd");       // location of first occurrence within s1
int pos = s1.find_first_of("cd", 10);   // location of first occurrence within s1 from char
                                        // 10 onwards
s2 = s1.substr(3, 5);                   // substring of s1 from pos 3 span 5 chars
s1.compare(s2);                         // Returns 0 if s1 and s2 are the same, less than 0
                                        // if s1<s2, greater than 0 if s1>s2.
```

Converting between c-string and string class

```
char *s1;           // c-style string
string s2;          // class based string
s1 = &s2[0];        // Convert class string to c-string by setting pointer s1 to address of
                    // character 0 of s2
s2 = s1;            // Convert c-string back to class string via overloaded = operator
```

Functions

- Functions parameters are passed by value by default.
- Functions must return a value. The return value need not be used.
- Procedures or valueless functions return 'void'.
- There must always be a main function that returns an int.
- Function declarations are never nested.
- Function overloading is available through use of different parameter signatures.

Normal, pass by value, function

```
double areaTriangle(double sideA, double sideB, double angleC) {
    const double pi = 3.14159265358979323846;
    double angleRadians = angleC * pi / 180; // Convert degrees to radians
    return 0.5 * sideA * sideB * sin(angleRadians);
}

int main() {
    double a, b, c;
    cout << "Area of a triangle calculator\n";
    cout << "Enter length of sideA, length of sideB and angle C in degrees: ";
    cin >> a >> b >> c;
    double area = areaTriangle(a, b, c); // Call our function
    cout << "The area of your triangle is:" << area << "\n";
    return 0;
}
```

Pass by reference

Use this in C++ instead of using pointers

```
int addTen(int& i) {     // receives i as a reference
    i = i + 10;
    return 0;
}

int main() {
    int i = 10;
    addTen(i);           // passes i by reference since addTen() receives it
                         // by reference, but the syntax makes it appear as if i is passed
                         // by value. The only way to know is to check the func definition.
    cout << i << "\n";
    return 0;
}
```

Main function signature

Program arguments may be accessed as strings through main's array argv with argc elements. First element will be the program name.

```
int main(int argc, char **argv)     // method 1
int main(int argc, char *argv[])    // method 2... they are equiv
```

Arrays

C style arrays

```
int numbers[100];                       // Allocate 100 integers
double stuff[200];                      // Allocate 200 doubles
int primes[8] = {2,3,4,5,7,11,13,17};   // Allocate and assign initial values
char* str = "This is a string";         // Character array with initial value
```

Multi-dimensional arrays

```
int a[10];      // a is a 10 int array. a[0] is first element. a[9] is last
char b[];       // for use in a function header, b is array of chars unknown length
int c[2][3];    // an array containing 2 arrays of 3 ints each. a[1][0] follows a[0][2]
```

An array without an index is a pointer to the array block (ie: its address)

```
int a[10], b[20];    // two arrays
int *p = a;          // p points to first int of array a
p = b;               // p now points to the first int of array b
```

An array or pointer with an index n in square brackets returns the nth value:

```
int a[10];           // an array of 10 ints
int *p;              // a pointer for integers
int i = a[0];        // i is the first element of a
i = *a;              // pointer dereference. i is the first element of a
p = a;               // same as p = &a[0]
p++;                 // same as p = p+1; same as p=&a[1]; same as p = a+1
```

- **Array bounds are not checked**; it is your responsibility to not past the end. Don't assume!
- Dereferencing a pointer: Use the `*` in front of the pointer name, eg, `` `int val = *ptr;` ``
- For looping through arrays, there is no `.length` property. You need to know the size.
  Usual practice is to declare a const for this eg `const int SIZE = 10;`,
  but you can calculate it if need be `int size = sizeof a / sizeof a[0];`

C++ 11 has a wrapper class array that provides additional functionality at no performance cost.

```
#include <array>
array<int, 10> primes = {1, 2, 3, 5, 7, 11, 13, 17, 19, 23};
int length = primes.size();
```

(HKOI doesn't tend to use this wrapper class)

Namespaces

The “using namespace” operands can be used to import functionality into the local space, removing the need for the “`namespace::`” prefix. Compare the use of *cout* in the two examples.

With localising namespace	Without localising namespace

`#include <iostream>`
`using namespace std;`
 	
`int main () {`
`  cout << "Hello World! ";`
`}`
	`#include <iostream>`
	
	`int main() {`
	`  std::cout << "Hello World!";`
	`}`

You can define your own namespaces to avoid naming collisions.

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

namespace foo {
  int myFunction() {
    return 5;
  }
}

namespace bar {
  const double pi = 3.1416;
  double myFunction() {
    return 2*pi;
  }
}

int main () {
  cout << foo::myFunction() << '\n';
  cout << bar::myFunction() << '\n';
  cout << bar::pi << '\n';
  return 0;
}
```

Classes

Access modifiers

- By default, all members of a class declared with the class keyword have private access for all its members.
- private members of a class are accessible only from within other members of the same class (or from their "friends").
- protected members are accessible from other members of the same class (or from their "friends"), but also from members of their derived classes.
- public members are accessible from anywhere where the object is visible.

Functions/methods:

- Can be defined inside the class definition or outside

```
#include <iostream>

using namespace std;

class Rectangle {
    // Private by default but never hurts to specify
    private:
        int width;
        int height;
    public:
        // Constructor
        Rectangle(int w, int h) {
            cout << "Rectangle constructor\n";
            width = w;
            height = h;
        }
        // Declare a method with its code
        int getArea() {
            return width * height;
        }
        // Declare a method without its code (for addition later)
        void updateWidth(int width);
};      // end with ;

// Add code to a method later this way
void Rectangle::updateWidth(int width) {
    // Use the class namespace prefix to resolve name collisions
    Rectangle::width = width;
};

int main() {
    Rectangle r (10, 5);
    cout << "Area of the rectangle: " << r.getArea() << "\n";
    r.updateWidth(20);
    cout << "Area of the rectangle: " << r.getArea() << "\n";
}
```

```
class Square: public Rectangle {   // Square inherits from Rectangle
    private:
        int side;
    public:
        // Passing along parameters from derived constructor to base constructor
        Square(int side):Rectangle(side, side) {
            Square::side = side;
            cout << "Square constructor\n";
        }
        int getPerimeter() {
            return (4*side);
        }
};

int main() {
    Square s (50);
    cout << "Area of the square: " << s.getArea() << "\n";
    cout << "Perimeter of the square: " << s.getPerimeter() << "\n";
}
```

http://www.cplusplus.com/doc/tutorial/classes/
http://www.cplusplus.com/doc/tutorial/templates/

Files

Include the following to work on files

- **ofstream**: Stream class to write on files
- **ifstream**: Stream class to read from files
- **fstream**: Stream class to both read and write from/to files.

Read a text file

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open()) {
    while ( getline (myfile,line) ) {
      cout << line << '\n';
    }
    myfile.close();
  } else cout << "Unable to open file";
  return 0;
}
```

Read entire text file to a string

```
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>

using namespace std;

string readFile(string filename) {
    // https://stackoverflow.com/a/19922123
    string out;
    stringstream buffer;
    ifstream f(filename);       // Open file
    if (f.is_open()) {
        buffer << f.rdbuf();    // Read file into stringstream buffer
        out = buffer.str();     // Copy buffer to string, `out`
        f.close();
    }
    return out;
}

int main() {
    string content = readFile("wild-wild-west.txt");
}
```

Write to a text file

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile ("example.txt");
  if (myfile.is_open()) {
    myfile << "This is a line.\n";
    myfile << "This is another line.\n";
    myfile.close();
  }
  else cout << "Unable to open file";
  return 0;
}
```

Read from a binary file

```
// reading an entire binary file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  streampos size;
  char * memblock;

  ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
  if (file.is_open())
  {
    size = file.tellg();
    memblock = new char[size];
    file.seekg(0, ios::beg);
    file.read(memblock, size);
    file.close();

    cout << "the entire file content is in memory";

    delete[] memblock;
  }
  else cout << "Unable to open file";
  return 0;
}
```

- The file is opened with the **`ios::ate`** flag, which means “at the end”, so the pointer will be positioned at the end of the file. Then, call **`tellg()`** to directly obtain the size of the file.

Seek functions to move about binary files

```
seekg ( position );       // Move get pointer to this absolute position (0=start of file)
seekp ( position );       // Move put pointer to this absolute position (0=start of file)

// for the next two use offset of ios::beg, ios::cur or ios::end
seekg ( offset, direction );     // Move get pointer to in this relative direction
seekp ( offset, direction );     // Move put pointer to in this relative position
```

Boolean settings available in the ios namespace:

```
ios::in      // Open for input operations.
ios::out     // Open for output operations.
ios::binary  // Open in binary mode.
ios::ate     // Set position to end of file (else if new, beginning of file)
ios::app     // All output operations are append to end of file
ios::trunc   // If file is output and it already exists, all content is deleted and replaced
ios::beg     // Offset counted from the beginning of the stream
ios::cur     // Offset counted from the current position
ios::end     // Offset counted from the end of the stream
```

State of the stream

```
bad()        // True if reading or writing operation fails
fail()       // True if bad(), or if a casting format error happens (like integer expected)
eof()        // True if end of file reached.
good()       // True if none of the above would be true.
```

Flush your stream (clear the buffer to disk)

```
yourfile.flush()
```

http://www.cplusplus.com/doc/tutorial/files/

Vectors

Vectors are sequence containers representing arrays that can change in size.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    // Create empty vector
    vector<int> values = {};

    // Read into the vector
    for (int i=0; i < 10; i++) {
        int tmp;
        cout << "\nEnter value " << (i+1) << ": ";
        cin >> tmp;
        values.push_back(tmp);
    }
    cout << "\n";

    // Iterate through the vector
    cout << "There are " << values.size() << " items in the vector\n";
    for (int i=0; i<values.size(); i++) {
        cout << "\nItem " << (i+1) << ": " << values[i];
    }

    // Pop off the vector
    cout << "\nPopping...\n";
    while (! values.empty()) {
        int tmp = values.back(); // Get reference to last element - use .front() for first
        values.pop_back(); // Remove last element
        cout << "\nPopped: " << tmp;
    }
}
```

Refer to http://www.cplusplus.com/reference/vector/vector/

Header files

circle.h - provide class and function definitions

```
#pragma once            // ensures is only included once
class Circle {
    private:
        double radius;
    public:
        Circle(double r);
        double getArea();
        double getCircumference();
};
```

circle.cpp - provide logic for your class methods and functions

```
#include <cmath> // Contains M_PI
#include "circle.h"

using namespace std;

Circle::Circle(double r) {
    radius=r;
}

double Circle::getArea() {
    return M_PI * radius * radius;
}

double Circle::getCircumference() {
    return 2 * M_PI * radius;
}
```

main.cpp

```
#include <iostream>
#include <cmath>       // Contains M_PI
#include "circle.h"    // Include our circle code
using namespace std;

int main() {
    Circle c(10);
    cout << "Area of circle: " << c.getArea() << "\n";
    cout << "Circumference of circle: " << c.getCircumference() << "\n";
}
```

Compile

```
g++ -o demo circle.cpp main.cpp
```

Some legacy C that might be useful

Dynamic memory

```
#include <cstdlib>
```

malloc - memory allocate
Syntax

```
casttype* ptr = (casttype*) malloc(byte-size);
```

Example

```
// Size of int is 4 bytes, therefore allocate 400 bytes of memory. ptr is address of the
first byte.
int* ptr = (int*) malloc(100 * sizeof(int));
```

free(p)

- free memory pointed at p; must have been alloc'd; don't re-free

calloc(n,s) - contiguous allocation

- Will clear/initialize the space to zeros;
- If space is insufficient, allocation fails and returns a NULL pointer.
- Example... This statement allocates contiguous space in memory for 25 elements each with the size of the float.

```
ptr = (float*) calloc(25, sizeof(float));
// this is equivalent to: float ptr[25];
```

Pointers

Pointers are indicated by left associative asterisk (*) in the type declarations:

```
int *a;        // a is a pointer to an integer
char *b;       // b is a pointer to a character
int *c[2];     // c is an array of two pointers to ints (same as int *(c[2]);
int (*d)[2];   // d is a pointer to an array of 2 integers
int *p = a;    // p points to first int of array a
```

- Pointers are simply integers holding memory addresses. Pointer variables may be assigned.
- Adding 1 computes pointer to the next value by adding sizeof(X) for type X
- Dereferencing the pointer: When you want to access the data/value in the memory that the pointer points to.
- General int adds to pointer (even 0 or negative values) behave in the same way
- Addresses may be computed with the ampersand (&) operator.

Struct

You can group values together in repeating sequences using arrays or in mixed groups called structs that contain a sequence of variables structured as indicated.

```
struct mydata {         // Define struct type mydata
   int count;
   double items[100];
};

struct mydata bob;      // Create a struct of type mydata of name bob

struct person {         // Define struct type person
   int age;
   char *name;
};

struct person alice = { 15, "Alice" };

printf("Person %s is %d years old\n", alice.name, alice.age);
```

- Dereferencing a pointer for struct: Use an arrow `->` to dereference a pointer to a field. Using above example, `bob->count` or `bob->items`
- A struct without a field is a pointer.

If a is a struct and b is a field, then...

a.b	Get member b of object a	
a->b	Get member b of object pointed to by a	equivalent to (*a).b
*a.b	Get value pointed to by member b of object a	equivalent to *(a.b)

Enumerated types

```
// Define a datatype days that can have these possible values....
typedef enum {
  SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
} days;

// Declare a variable of that type, set initial value
days today = TUESDAY;

if (today == WEDNESDAY) {
   printf("It is Wednesday\n");
}
```

- Actually uses integers “under the hood”, so in above case SUNDAY == 0 and MONDAY == 1 etc