
# CODEMATES

# C++ CHEATSHEET

## (Preview)

**LITERALS:-** C++ Constants/Literals- Constants refer to fixed values that the program may not alter and they are called literals. Constants can be of any of the basic data types and can be divided into Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.

```
2125, 0377, 0xff                // Integers (decimal, octal, hex)
3543.0, 1.13e2                  // double (real) numbers
'a', '\191', '\x71'             // Character (literal, octal, hex)
true, false                     // bool constants 1 and 0
'\n', '\\', '\'', '\"'          // Newline, backslash, single quote, double quote
"strings\n"                     // Array of characters ending with newline and \0
 "code" "mates"                 // Concatenated strings
2147483647L, 0x7fffffffl        // Long (32-bit) integers
```

## VARIABLE DECLARATIONS AND INITIALIZATIONS:-

A variable provides us with named storage that our programs can manipulate. Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

```
int x;                  // Declare x to be an integer (value undefined)
```

```
int x=255;                // Declare and initialize x to 255
short s; long l;          // Usually 16 or 32 bit integer (int may be either)
char c='a';               // Usually 8 bit character
unsigned char u=255; signed char s=-1;   // char might be either
unsigned long x=0xffffffffL;            // short, int, long are signed
float f; double d;        // Single or double precision real (never unsigned)
bool b=true;              // true or false, may also use int (1 or 0)
int a, b, c;              // Multiple declarations
int a[10];                // Array of 10 integers (a[0] through a[9])
int a[]={0,1,2};          // Initialized array (or a[3]={0,1,2}; )
int a[2][3]={{1,2,3},{4,5,6}};  // Array of array of integers
char s[]="hello";         // String (6 elements including '\0')
int* p;                   // p is a pointer to (address of)
int char* s="hello";          // s points to unnamed array containing "hello"
void* p=NULL;             // Address of untyped memory (NULL is 0)
int& r=x;                 // r is a reference to (alias of) int x
enum weekend {SAT,SUN};   // weekend is a type with values SAT and SUN
enum weekend day;         // day is a variable of type weekend
enum weekend {SAT=0,SUN=1};  // Explicit representation as int
enum {SAT,SUN} day;       // Anonymous enum
typedef String char*;     // String s; means char* s;
const int c=3;            // Constants must be initialized, cannot assign to const
int* p=a;           // Contents of p (elements of a) are constant
int* const p=a;           // p (but not contents) are constant const
int* const p=a;     // Both p and its contents are constant const
int& qw=x;          // qw cannot be assigned to change x
```

**STATEMENTS:-** **Statements are fragments of the C++ program that are executed in sequence. The body of any function is a sequence of statements. C++ includes the following types of statements:**

**1) expression statements;**
**2) compound statements;**
**3) selection statements;**
**4) iteration statements;**
**5) jump statements;**
**6) declaration statements;**
**7) try blocks;**
**8) atomic and synchronized blocks**

```
x=y;                    // Every expression is a statement

int x;                  // Declarations are statements

 ;                      // Empty statement

{                       // A block is a single statement

int x;                  // Scope of x is from declaration to end of block

 a;                     // In C, declarations must precede statements

}

if (x) a;               // If x is true (not 0), evaluate a

else if (y) b;          // If not x and y (optional, may be repeated)

else c;                 // If not x and not y (optional)

while (x) a;            // Repeat 0 or more times while x is true

 for (x; y; z) a;       // Equivalent to: x; while(y) {a; z;}

do a; while (x);        // Equivalent to: a; while(x) a;

switch (x) {            // x must be int

case X1: a;             // If x == X1 (must be a const), jump here

case X2: b;             // Else if x == X2, jump here

default: c;             // Else jump here (optional) }

break;                  // Jump out of while, do, or for loop, or switch

continue;               // Jump to bottom of while, do, or for loop

return x;               // Return x from function to caller

 try { a; }

catch (T t) { b; }      // If a throws a T, then jump here

catch (...) { c; }      // If a throws something else, jump h
```

**EXPRESSIONS:-** C++ expression consists of operators, constants, and variables which are arranged according to the rules of the language. It can also contain function calls which return values.

Operators are grouped by precedence, highest first. Unary operators and assignment evaluate right to left. All others are left to right. Precedence does not affect order of evaluation, which is undefined. There are no run time checks for arrays out of bounds, invalid pointers, etc.

```
x * y                 // Multiply
 x / y                // Divide (integers round toward 0)
 x % y                // Modulo (result has sign of x)
 x + y                // Add, or &x[y]
x - y                 // Subtract, or number of elements from *x to *y
++x                   // Add 1 to x, evaluates to new value (prefix)
--x                   // Subtract 1 from x, evaluates to new value
 ~x                   // Bitwise complement of x
 !x                   // true if x is 0, else false (1 or 0 in C)
 -x                   // Unary minus
+x                    // Unary plus (default)
&x                    // Address of x
T::X                  // Name X defined in class T
N::X                  // Name X defined in namespace
 N ::X                // Global name X
 t.x                  // Member x of struct or class t
p->x                  // Member x of struct or class pointed to by p
a[i]                  // i'th element of array a
f(x,y)                // Call to function f with arguments x and y
T(x,y)                // Object of class T initialized with x and y
x++                   // Add 1 to x, evaluates to original x (postfix)
x--                   // Subtract 1 from x, evaluates to original x
typeid(x)             // Type of x
typeid(T)             // Equals typeid(x) if x is a T
dynamic_cast<T>(x)    // Converts x to a T, checked at run time
```

```
 static_cast<T>(x)        // Converts x to a T, not checked
 reinterpret_cast<T>(x)   // Interpret bits of x as a T
const_cast<T>(x)          // Converts x to same type T but not const
sizeof x                  // Number of bytes used to represent object x
sizeof(T)                 // Number of bytes to represent type T
*p                        // Contents of address p (*&x equals x)
new T                     // Address of newly allocated T object
new T(x, y)               // Address of a T initialized with x, y
 new T[x]                 // Address of allocated n-element array of T
delete p                  // Destroy and free object at address p
delete[] p                // Destroy and free array of objects at p
(T) x                     // Convert x to T (obsolete, use .._cast<T>(x))
 x << y                   // x shifted y bits to left (x * pow(2, y))
x >> y                    // x shifted y bits to right (x / pow(2, y))
x < y                     // Less than
x <= y                    // Less than or equal to
 x > y                    // Greater than
x >= y                    // Greater than or equal to
x == y                    // Equals
x != y                    // Not equals
x & y                     // Bitwise and (3 & 6 is 2)
x ^ y                     // Bitwise exclusive or (3 ^ 6 is 5)
x | y                     // Bitwise or (3 | 6 is 7)
x && y                    // x and then y (evaluates y only if x (not 0))
x || y                    // x or else y (evaluates y only if x is false (0))
x = y                     // Assign y to x, returns new value of x
x += y                    // x = x + y, also -= *= /= <<= >>= &= |= ^=
x ? y : z                 // y if x is true (nonzero), else z
throw x                   // Throw exception, aborts if not caught
x , y                     // evaluates x and y, returns y (seldom used)
```

THIS IS JUST A PREVIEW WHICH HAS SOME TOPICS COVERED. IN OUR FULL CHEATSHEET WE HAVE COVERED MOST OF IMPORTANT C++ TOPICS CONCEPTS. TO GET ACCESS TO IT FOLLOW US AND SEND DM. WE WILL SEND YOU LINK TO OUR FULL CHEATSHEET PERSONALLY.

JUST A SINGLE FOLLOW CAN HELP YOU IN LAST MOMENTS OF YOUR REVISION BEFORE AN INTERVIEW OR  UNIVERSITY EXAM. FOR MORE CHEATSHEETS BE WITH US. TO CHECK WHAT YOU CAN GET MORE SEE NEXT PAGE.

## FUNCTIONS:-

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

Function parameters and return values may be of any type. A function must either be declared or defined before it is used. It may be declared first and defined later. Every program consists of a set of a set of global variable declarations and a set of function definitions (possibly in separate files), one of which must be:

```
int main()  { statements... }     or int main(int argc, char* argv[]) { statements... }
```

argv is an array of argc strings from the command line. By convention, main returns status 0 if successful, 1 or higher for errors.

Functions with different parameters may have the same name (overloading). Operators except :: . .* ?: may be overloaded. Precedence order is not affected. New operators may not be created.

```
int f(int x, int);        // f is a function taking 2 integers and returning

int void f();               // f is a procedure taking no arguments

void f(int a=0);          // f() is equivalent to f(0)

f();                    // Default return type is int

 inline f();              // Optimize for speed

f() { statements; }       // Function definition (must be global)

T operator+(T x, T y);    // a+b (if type T) calls operator+(a, b)

 T operator-(T x);         // -a calls function operator-(a)
```

## CLASSES:-

A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object. For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

- A Class is a user defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behaviour of the objects in a Class.
- In the above example of class *Car*, the data member will be *speed limit*, *mileage* etc and member functions can be *apply brakes*, *increase speed* etc.

```
class T {               // A new type

private:                // Section accessible only to T's member

functions protected:              // Also accessible to classes derived from T

public:                 // Accessible to all

int x;                // Member data

void f();             // Member function

void g() {return;}      // Inline member function

void h() const;         // Does not modify any data members

int operator+(int y);   // t+y means t.operator+(y)

int operator-();        // -t means t.operator-()   T

(): x(1) {}           // Constructor with initialization list

T(const T& t): x(t.x) {}  // Copy constructor
```

```
class X: public virtual T {}; // Classes derived from X have base T
```

**TEMPLATES:-** Templates are powerful features of C++ which allows you to write generic programs. In simple terms, you can create a single function or a class to work with different data types using templates. Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs.

```
template <class T> T f(T t);        // Overload f for all types
 template <class T> class X {        // Class with type parameter T
 X(T t); };                          // A constructor
template <class T> X<T>::X(T t) {}  // Definition of constructor
X<int> x(3);                         // An object of type "X of int"
template <class T, class U=T, int n=0>  // Template with default parameters
```

## C/C++ STANDARD LIBRARY:-

The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc. It is a library of container classes, algorithms, and iterators. It is a generalized library and so, its components are parameterized. A working knowledge of template classes is a prerequisite for working with STL.

# SOME IMPORTANT AND MOST USED STL FUNCTIONS

## STRING - Variable sized character array

```
string s1, s2="hello";    // Create strings
 s1.size(), s2.size();     // Number of characters: 0, 5
s1 += s2 + ' ' + "world"; // Concatenation
s1 == "hello world"       // Comparison, also <, >, !=, etc.
s1[0];                    // 'h'
 s1.substr(m, n);          // Substring of size n starting at s1[m]
s1.c_str();               // Convert to const char*
getline(cin, s);          // Read line ending in '\n'
```

## VECTOR- Variable sized array/stack with built in memory allocation

```
vector<int> a(10);        // a[0]..a[9] are int (default size is 0)
a.size();                 // Number of elements (10)
```