C++ Fundamentals Cheat Sheet

Table of Contents

Basic Syntax & Data Types

Program Structure

cpp

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

int main() {
    // Your code here
    return 0;
}
```

Data Types

cpp

```cpp
// Primitive Types
int num = 42;                // 4 bytes
long longNum = 1000000L;     // 8 bytes
float decimal = 3.14f;       // 4 bytes
double precise = 3.14159;    // 8 bytes
char letter = 'A';           // 1 byte
bool flag = true;            // 1 byte
string text = "Hello";       // Variable size

// Constants
const int MAX_SIZE = 100;
#define PI 3.14159
```

Input/Output

cpp

```cpp
cout << "Hello World" << endl;
cin >> variable;
getline(cin, stringVariable);
```

Control Structures

Conditionals

cpp

```cpp
// If-else
if (condition) {
    // code
} else if (condition2) {
    // code
} else {
    // code
}

// Switch
switch (variable) {
    case 1:
        // code
        break;
    case 2:
        // code
        break;
    default:
        // code
}

// Ternary operator
result = (condition) ? value1 : value2;
```

Loops

cpp

```cpp
// For loop
for (int i = 0; i < n; i++) {
    // code
}

// While loop
while (condition) {
    // code
}

// Do-while loop
do {
    // code
} while (condition);

// Range-based for loop (C++11)
for (auto element : container) {
    // code
}
```

Functions

Basic Function Syntax

cpp

```cpp
// Function declaration
returnType functionName(parameters);

// Function definition
int add(int a, int b) {
    return a + b;
}

// Function with default parameters
void greet(string name = "World") {
    cout << "Hello " << name << endl;
}

// Function overloading
int multiply(int a, int b);
double multiply(double a, double b);

// Pass by reference
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Arrays & Strings

Arrays

cpp

```cpp
// Declaration and initialization
int arr[5] = {1, 2, 3, 4, 5};
int arr2[] = {1, 2, 3};  // Size inferred

// Multidimensional arrays
int matrix[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};

// Array operations
int size = sizeof(arr) / sizeof(arr[0]);
```

Strings

cpp

```cpp
#include <string>

string str = "Hello";
string str2("World");

// String methods
str.length()           // Get length
str.size()             // Same as length()
str.empty()            // Check if empty
str.substr(pos, len)   // Get substring
str.find("text")       // Find substring
str.replace(pos, len, "new")  // Replace
str += " World";       // Concatenation
str[0] = 'h';          // Access character
```

Pointers & References

Pointers

cpp

```cpp
int x = 10;
int* ptr = &x;         // Pointer declaration
*ptr = 20;             // Dereference
cout << ptr;           // Address
cout << *ptr;          // Value

// Null pointers
int* nullPtr = nullptr;  // C++11
int* nullPtr2 = NULL;    // Older style
```

References

cpp

```cpp
int x = 10;
int& ref = x;          // Reference must be initialized
ref = 20;              // Changes x to 20
// References cannot be reassigned
```

Object-Oriented Programming

Class Definition

cpp

```cpp
class ClassName {
private:
    int privateVar;

protected:
    int protectedVar;

public:
    int publicVar;

    // Constructor
    ClassName(int val) : privateVar(val) {}

    // Destructor
    ~ClassName() {}

    // Methods
    void setVar(int val) { privateVar = val; }
    int getVar() const { return privateVar; }

    // Static members
    static int staticVar;
    static void staticMethod() {}
};
```

Inheritance

cpp

```cpp
class Base {
public:
    virtual void virtualMethod() {}
    virtual ~Base() {}  // Virtual destructor
};

class Derived : public Base {
public:
    void virtualMethod() override {}  // C++11
    // or: void virtualMethod() {}    // Older style
};
```

Key OOP Concepts

cpp

```cpp
// Encapsulation: Private/protected members
// Inheritance: class Derived : public Base
// Polymorphism: Virtual functions
// Abstraction: Pure virtual functions
class Abstract {
public:
    virtual void pureVirtual() = 0;  // Pure virtual
};
```

STL Containers

Vector

cpp

```cpp
#include <vector>

vector<int> v;                    // Empty vector
vector<int> v2(5);                // Size 5, default values
vector<int> v3(5, 10);            // Size 5, all elements = 10
vector<int> v4 = {1, 2, 3, 4, 5}; // Initializer list

// Methods
v.push_back(element);             // Add to end
v.pop_back();                     // Remove from end
v.size();                         // Get size
v.empty();                        // Check if empty
v.clear();                        // Remove all elements
v.front();                        // First element
v.back();                         // Last element
v[index];                         // Access by index
v.at(index);                      // Safe access with bounds checking
v.insert(iterator, value);        // Insert at position
v.erase(iterator);                // Remove at position
```

Map (Hash Table)

cpp

```cpp
#include <map>
#include <unordered_map>

// Ordered map (Red-Black Tree)
map<string, int> m;
m["key"] = value;                 // Insert/update
m.insert({"key", value});         // Insert
m.find("key");                    // Find (returns iterator)
m.count("key");                   // Check existence (0 or 1)
m.erase("key");                   // Remove
m.size();                         // Get size
m.empty();                        // Check if empty

// Unordered map (Hash Table) - Better performance
unordered_map<string, int> um;
// Same methods as map
```

Set

cpp

```cpp
#include <set>
#include <unordered_set>

set<int> s;                     // Ordered set
s.insert(value);                // Insert
s.erase(value);                 // Remove
s.find(value);                  // Find
s.count(value);                 // Check existence
s.size();                       // Get size
s.empty();                      // Check if empty

unordered_set<int> us;          // Hash set (better performance)
```

Queue & Stack

cpp

```cpp
#include <queue>
#include <stack>

// Queue (FIFO)
queue<int> q;
q.push(element);                // Add to back
q.pop();                        // Remove from front
q.front();                      // Access front
q.back();                       // Access back
q.size();                       // Get size
q.empty();                      // Check if empty

// Stack (LIFO)
stack<int> st;
st.push(element);               // Add to top
st.pop();                       // Remove from top
st.top();                       // Access top
st.size();                      // Get size
st.empty();                     // Check if empty
```

Priority Queue

cpp

```cpp
#include <queue>

priority_queue<int> pq;          // Max heap by default
priority_queue<int, vector<int>, greater<int>> minPq; // Min heap

pq.push(element);                // Insert
pq.pop();                        // Remove top
pq.top();                        // Access top
pq.size();                       // Get size
pq.empty();                      // Check if empty
```

Memory Management

Dynamic Memory

cpp

```cpp
// C++ style (recommended)
int* ptr = new int(10);          // Allocate single int
int* arr = new int[5];           // Allocate array
delete ptr;                      // Free single object
delete[] arr;                    // Free array

// Smart pointers (C++11) - Automatic memory management
#include <memory>
unique_ptr<int> uptr = make_unique<int>(10);
shared_ptr<int> sptr = make_shared<int>(10);
```

File I/O

Basic File Operations

cpp

```cpp
#include <fstream>

// Writing to file
ofstream outFile("filename.txt");
outFile << "Hello World" << endl;
outFile.close();

// Reading from file
ifstream inFile("filename.txt");
string line;
while (getline(inFile, line)) {
    cout << line << endl;
}
inFile.close();

// Both read and write
fstream file("filename.txt", ios::in | ios::out);
```

Common Algorithms

Essential STL Algorithms

cpp

```cpp
#include <algorithm>

// Sorting
sort(v.begin(), v.end());         // Ascending
sort(v.begin(), v.end(), greater<int>()); // Descending

// Searching
find(v.begin(), v.end(), value); // Linear search
binary_search(v.begin(), v.end(), value); // Binary search (sorted)

// Min/Max
*min_element(v.begin(), v.end());
*max_element(v.begin(), v.end());

// Reverse
reverse(v.begin(), v.end());

// Count
count(v.begin(), v.end(), value);
```

Quick Reference - Time Complexities

Operation	Vector	Map	Unordered_Map	Set	Unordered_Set
Insert	O(1)*	O(log n)	O(1) avg	O(log n)	O(1) avg
Delete	O(n)	O(log n)	O(1) avg	O(log n)	O(1) avg
Search	O(n)	O(log n)	O(1) avg	O(log n)	O(1) avg
Access	O(1)	O(log n)	O(1) avg	O(log n)	O(1) avg

*O(n) when reallocation occurs

Compilation Commands

bash

```bash
g++ -o program program.cpp        # Basic compilation
g++ -std=c++11 -o program program.cpp  # C++11 standard
g++ -Wall -Wextra -o program program.cpp  # With warnings
```