C++ Cheatsheet

Basics

Boilerplate

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

int main() {
    cout << "Welcome To Pradeep Kumar";
    return 0;
}
```

Input/Output

- **Output (`cout <<`)** – Prints to console
- **Input (`cin >>`)** – Takes user input

```cpp
string name;
cout << "Enter name: ";
cin >> name;
cout << "Hello, " << name;
```

Data Types

C++ supports several data types, divided into categories:

Type	Example	Size (Typical)	Description
`char`	`char c='A';`	1 byte	Stores single character
`int`	`int x=10;`	4 bytes	Integer numbers
`short`	`short s=100;`	2 bytes	Small integer
`long`	`long l=100000;`	4-8 bytes	Large integer
`float`	`float f=10.5;`	4 bytes	Single-precision decimal
`double`	`double d=10.55;`	8 bytes	Double-precision decimal
`bool`	`bool flag=true;`	1 byte	True/False values
`void`	–	–	Represents no value

Escape Sequences

Sequence	Meaning
`\a`	Alert (beep)
`\b`	Backspace
`\f`	Form feed
`\n`	Newline
`\r`	Carriage return
`\t`	Horizontal tab
`\\`	Backslash
`\'`	Single quote

Sequence	Meaning
`\"`	Double quote
`\?`	Question mark
`\0`	Null terminator
`\nnn`	Octal value
`\xhh`	Hexadecimal value

Comments

```
// Single-line comment

/* Multi-line
   comment */
```

Strings

```
#include <string>

string s1 = "Hello";
string s2 = "World";

// Concatenation
string s3 = s1 + " " + s2;

// Append
s1.append(s2);

// Length
cout << s3.length();
```

```
// Access/modify
s3[0] = 'h';
```

Math Functions (`<cmath>`)

Function	Example	Result
`max(a,b)`	`max(5,10)`	10
`min(a,b)`	`min(5,10)`	5
`sqrt(x)`	`sqrt(144)`	12
`ceil(x)`	`ceil(1.9)`	2
`floor(x)`	`floor(1.9)`	1
`pow(x,y)`	`pow(2,3)`	8
`abs(x)`	`abs(-5)`	5
`round(x)`	`round(2.6)`	3

Decision Making

```
if (x > 0) { ... }
else if (x == 0) { ... }
else { ... }

// Ternary
int result = (x > 0) ? 1 : -1;

// Switch
switch (choice) {
    case 1: cout << "One"; break;
    case 2: cout << "Two"; break;
```

```
    default: cout << "Other";
}
```

Loops

```
// While loop
while (i < 5) { i++; }

// Do-while loop
do { i++; } while (i < 5);

// For loop
for (int i=0; i<5; i++) { ... }

// Break/Continue
break;     // exits loop
continue;  // skips current iteration
```

References

```
int a = 10;
int &ref = a;
ref = 20; // a becomes 20
```

Pointers

```
int x = 10;
int *ptr = &x;    // Pointer to x
cout << *ptr;     // Dereference (prints 10)
```

- `nullptr` is used for null pointers in C++11+
- Use `->` operator to access members of objects through pointers

Functions & Recursion

```
int add(int a, int b) {
    return a + b;
}

// Recursion
int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n-1);
}
```

Object-Oriented Programming

Class & Object

```
class Car {
public:
    string brand;
    int year;
    void drive() { cout << "Driving"; }
};
```

```
Car c1;
c1.brand = "BMW";
c1.drive();
```

Constructor

```
class Car {
public:
    Car(string b, int y) {
        brand = b;
        year = y;
    }
    string brand;
    int year;
};
Car c("Audi", 2023);
```

Inheritance

```
class Vehicle {
public:
    void honk() { cout << "Beep!"; }
};

class Car : public Vehicle { };

Car obj;
obj.honk(); // Inherited method
```

Polymorphism (Virtual Functions)

```
class Animal { public: virtual void sound() { cout<<"Some sound"; }};
class Dog : public Animal { public: void sound() override { cout<<"Bark"; }};
Animal* a = new Dog();
a->sound(); // Bark
```

File Handling

```
#include <fstream>
ofstream myFile("test.txt"); // write
myFile << "Hello";
myFile.close();

ifstream readFile("test.txt");  // read
string line;
while (getline(readFile, line)) { cout << line; }
readFile.close();
```

- **Modes:** `ios::in` , `ios::out` , `ios::app` , `ios::binary` , `ios::ate` , `ios::trunc`

Exception Handling

```
try {
    throw runtime_error("Error occurred");
}
catch (exception &e) {
    cout << "Caught: " << e.what();
}
```

Additional Key Concepts

- **Namespaces:** Avoids name conflicts

```
namespace A { int x = 10; }
cout << A::x;
```

- **Templates:** Generic programming

```
```

- **STL (Standard Template Library):** `vector`, `map`, `set`, `stack`, etc.
