C++ Cheat Sheet

C++ Cheat Sheet. Based on C++ Tutorial 2020 by Derek Banas

Fundamentals

Key Concepts and Syntax Before Creating Custom Classes

Recall	Notes
Comments	`//Single Line` `/* Multi-Line */`

Common Includes	`#include <cstdlib> //C STD Lib`
	`#include <iostream> //Read/Write`
	`#include <string> //Strings`
	`#include <limits> //Min Max`
	`#include <vector> //Vectors`
	`#include <sstream> //SStreams`
	`#include <numeric>`
	`#include <ctime>`
	`#include <cmath>`
	`#include <algorithm> //Sort,`
	`#include <fstream> //Files`

Stream Insertion Operator	`// << Stream Insertion Operator` `std::cout << "Hello" << std::endl;`

Namespace	`using namespace std;`
	`int main(int argc, char** argv) {`
	`cout << "Hello" << endl; //With Namespace std`
	`std::cout << "Hello" << std::endl; //W/O NS`
	`return 0;`
	`}`

Main Function	`int main(int argc, char** argv) {`
	`//argc = Number of Argumented To Be Passed Inside Main`
	`//argv = Array of Pointers To Strings`
	`//Return Int`
	`//Returns 0 When No Errors`
	`}`

Variables

```
int letter123_ = 0;//Variable Names Can be AlphaNumeric with underscores, but can't start with a
bool boy = True;// [T|F]
char grade = 'A';// [256 Character Possibilities]
// Int
unsigned short int a;//[0, 65535]
short int a; //[-32768,32767]
int a;//[-2147483648, 2147483647]
unsigned int a;//[0,4294967295]
long int a;//[-2147483648,2147483647]
unsigned long int a;//[0,4294967295]
//Same KeyWords Can be placed infront of
float a;
double a;
//Use numeric_limits<DataType>::max(); to Find Max or Min
numeric_limits<double>::max();
numeric_limits<double>::min();
```

Print

```
printf("%c %5d %.2f %s\n", '(',1,1.1211,"...HELLO)!");
std::cout << "Hello" << std::endl;
```

User Input

```
string q1 = "Enter Your Name:";
string userName;
cout << q1;
cin >> userName;
cout << "Hello " << userName;
```

String → Integer

```
int userAge = stoi("15");
```

If/Else Statement

```
if (boolA){//Do Code For BoolA = True
}else if (boolB){//Do Code For BoolB = True
}else{//Do Code For None Above Conditions
}
```

Ternary Operator

```
userAge = 7
bool amIAKid = (userAge<=13) ? true : false;
```

Arrays

```
//Use Arrays For Collections of Same Type That You Know A Max Size. Use Vectors For Collections
int arryNum[5]={1};//{5,,,,}
int arrNum[]={2,3,4};//{2,3,4}
int arrNum[5]={5,6};////{5,6,,,}
int arrNum[3][3][3] = {{{1,2,3},{4,5,6},{7,8,9}},{{11,12,13},{14,15,16},{17,18,19}},{{21,22,23},
```

Vectors

```
//Use Vectors For Collections of Same Type That You Do NOT Know A Max Size. Use Arrays For Colle
vector<int> vKidAges(2);
vKidAges[0] = 5;
vKidAges[1] = 4;
vKidAges.push_back(5);
```

Loops

```
while(boolCondition1){
  //Loop Code
  continue; //Use if you want to stop code after "continue" word and jump back to the top of whi
  break; //Use if you want to break/exit out of the loop.
}
for(int i=0; i<15; ++i){
  //Loop Code
}
int arryNum[] = {1,2,3};
for(int x:arryNum) cout << x; //Single Line For Loop
do{
```

```
//Code Will Run At Least One Time. Check's condition after Run
}while(boolCondition2)
```

Strings

```
string greeting = "Hello John";
char firstCharacter = greeting[0];//H
char lastCharacter = greeting.back();//N
int strLength = greeting.length();//10
string greetingCopy = greeting;//Hello John
string  personName(greeting,6);//John(String,StartPosition)
string greetingExclamation = greeting + "!";//Hello John!
greetingExclamation.append("!");//Hello John!!
greeting.erase(6,greeting.length()-1).append("Ron"); //Hello Ron
if (greeting.find("Ron")!=string::npos)
  cout << "String Index:" << greeting.find("Ron");//Starting Index:6 (SearchTerm)
string hell=greeting.substr(0,4);//Hell (Start,Length)
string eighteen =to_string(10+8);//18
bool isLetterOrNumber = isalnum('A');//True
bool isLetter = isalpha('A');//True
bool isNumber = isdigit('2');//True
bool isSpace = isspace(' ');//True
```

Math

https://en.cppreference.com/w/cpp/numeric/math

Functions

```
double AddNumbers(double num1, double num2);//Create Function Prototype.Inputs: double num1, num
int main(int argc, char** argv) {
   printf("%.1f + %.1f = %.1f",5.0,4.0,AddNumbers(5,4));
   return(0);
}
double AddNumbers(double num1, double num2){
  //Local Variables Are Lost When Function Finishes
  return num1 + num2;//Function Code
}
```

Pointers

```
//Pointers Store Memory Address. Have Type
void AssignGrade(int* pGrade);//Star Allows to Recieve A Pointer
void CurveTest(int* pArrTest, int size);
int main(int argc, char** argv) {
   int userGrade=65;
   AssignGrade(&userGrade);//Use "&" Sign to send Memory Address
   cout << userGrade; //96
   int JohnGrade = 78;
   int* pJohnGrade = NULL; //Declares A Pointer(Give NULL when declaring)
   pJohnGrade = &JohnGrade; //"&" To Pass Memory Address of JohnGrade
   cout << "Address:" << pJohnGrade <<endl;
   cout << "Value:" << *pJohnGrade << endl; //Dereference operator (*) to see value at Memory Ad
   int arrGrades[] = {95,97,85,99,65};
   int* pArrGrades = arrGrades;//Don't Need "&" For Arrays B/C Arrays Store Memory Address Alrea
   CurveTest(arrGrades,5);
   cout << "1st: "<<pArrGrades<<", Value: "<<*pArrGrades<<endl;//1st: 0x000000(Some Address), Va
   pArrGrades++;
   cout << "2nd: "<<pArrGrades<<", Value: "<<*pArrGrades<<endl;//2nd: 0x000000(Some Address), Va
   return(0);
}
void AssignGrade(int* pGrade){//Star Allows to Recieve A Pointer
   *pGrade=96;//Dereference operator (*) to change Value at Memory Address Given by pointer
}
void CurveTest(int* pArrTest, int size){
   for (int i=0; i<size;++i){
      pArrTest[i] = pArrTest[i] + 1;//Add 1 to Value
   }
}
```

Try/Catch

```
try {//Do Code
  throw "I Am An Exception";//Some Code Throws Exception
}catch (const char* exp){//Handle That Exception
  cout<<exp<<endl;
}
```

Quick Summary

All Code runs in main function which returns 0 when no errors. Use *using namespace* to easily access functions or variables with-in a certain namespace. Short, Long, Unsigned keywords can be added in front of integers, floats, or doubles. Global variables are defined outside the scope of a function, local variables are declared inside the scope of a function. Local variables are destroyed when the function finishes executing. *cout* with the stream insertion operator(<<) can be used to print output, while *cin* with the stream extraction operator (>>) can be used to receive input. *printf* can be used for formatted printing of characters, integers, doubles, and strings. If/Else Statements and the Ternary Operator can be used for conditional execution. While, For, and Do Loops can be used for iterations. Strings can be added, appended, erased, searched, or sub-stringed. Variables names can be alphanumeric with underscores, but can't start with a number. Arrays(*int arrA[3]*) are used for collections of a same type variable with some known maximum length, while vectors(*vector<int> vectA(3)*) are used for creating a collection of same type variable without a known maximum length or dynamically changing length. Vectors still need some arbitrary initial length. Pointers store the memory address of a variable and are declared with a * right after the type, *int* pVar=&referenceVariable*. The "&" is used to pass the memory address of the reference variable, *referenceVariable*. A dereference operator (*) is used to change value at memory address given by pointer and directly precedes the variable name, **pVar=5*. Arrays are pointers, so you don't use the "&" when make another pointer to the array. Functions other than main need function prototypes, which tell the compiler initialization information about the function such that they can be called before the body of the function is defined. A try/catch block can try to execute code and if exceptions are *thrown,* it can be handled in a catch block.

📌 **Pointers:** Store memory address and the dereference operator(*) is required to change the value at the memory address given by the pointer. Pointers that are declared without a specific memory address, should be initialized as NULL. "&" is used to pass the memory address of a variable.

C++ OOP and Essentials

Objects in real life have attributes and capabilities. Attributes → Fields. Capabilities → Methods. Classes list all fields and methods.

Recall	Notes

Header File	`//Contains Prototype Information`
	`#ifndef PERSON_H //Prevent Accidently Including Multiple Header Files`
	`#define PERSON_H`
	
	`class Person {`
	`protected:// Protected: Accessible in class and inherited Classes`
	`  int age;`
	`  char gender;`
	`  float cash;`
	`public:// Public: Accessible to anything that has access to the Object`
	`  static int numOfPeople; //Shared Value For Any Object of This Class Type`
	`  Person(int age, char gender);//Constructors called when object is created.`
	`  Person(int age, char gender, float cash);`
	`  Person();`
	`  virtual ~Person(); // Destructor called when object is destroyed.`
	`  void SetAge(int age);//Setters Used to Protect Data`
	`  void SetGender(char gender);`
	`  void SetCash(float cash);`
	`  int GetAge();`
	`  char GetGender();`
	`  float GetCash();`
	`  static int GetNumOfPeople();//Only static methods can access static fields`
	`  virtual double PersonInfo(); //Marked as virtual b/c each class that inherits from this will`
	
	`private:// Private: Accessible to only the class`
	
	`};`
	`#endif`

Implementation File

```
#include "Person.h" //Include Header File
Person::Person(int age, char gender) { //Constructors
    this->age = age;
    this->gender = gender;
    this->cash = 0.0;
}
Person::Person(int age, char gender, float cash) {
    this->age = age;
    this->gender = gender;
    this->cash = cash;
}
void Person::SetAge(int age){this->age = age;}
void Person::SetGender(char gender){this->gender = gender;}
void Person::SetCash(float cash){this->cash = cash;}
int Person::GetAge(){return age;}
char Person::GetGender(){return gender;}
float Person::GetCash(){return cash;}
int Person::GetNumOfPeople(){return numOfPeople;}
double Person::PersonInfo(){
    return 0.0;//Generic Person-Use Polymorphism, so similar object can be treated same. So, Stu
}
int Person::numOfPeople = 0;//Init Num of People
Person::~Person() = default;// Use default destructor
```

Inherit Header File

```
#ifndef STUDENT_H
#define STUDENT_H
class Student: public Person {//Inherits Person
public:
    Student();
    Student(int age, char gender);
    Student(const Student& orig);
    virtual ~Student();
    double PersonInfo();//Override PersonInfo. Will use Polymorphism to select right method
};
#endif
```

Inherit Implementation File

```
#include "Person.h"
#include "Student.h"
#include <cmath>
Student::Student(int age, char gender): Person(age, gender){ //Student Constructor Will Call Per
}
Student::~Student() = default;
double Student::PersonInfo(){//Different Method From Default.
    return 1.0;
}
```

Polymorphism

```
//Polymorphism allows a function to take a base class object as an input and call methods that m
#include <cstdlib>
#include <iostream>
#include "Person.h"
#include "Student.h"
using namespace std;
void ShowPersonInfo(Person& person);
int main(int argc, char** argv) {
    Person John(18,'M');
    Student Erin(19,'F');
    ShowPersonInfo(John);//Person Info:0
    ShowPersonInfo(Erin);//Person Info:1
    return 0;
}
void ShowPersonInfo(Person& person){//Polymorphism. Student inherits from Person. They both have
    cout << "Person Info:" << person.PersonInfo()<<endl;
}
```

```
//Abstract Class that is utilized for the purpose of being inheritted by other class. And not me
#include <iostream>
class Person{ //Abstract class
public:
    virtual double PersonInfo() = 0;//Marked as virtual b/c each class that inherits from this w
};
```

Abstract Class

```
class Student: public Person{//Inherits Person
protected:
    int age; char gender;
public:
    Student(int a, char g){age = a;gender = g;}
    double PersonInfo() override {//Should use Override KeyWord
        return 1.0;
    }
};
void ShowPersonInfo(Person& person);// Function Prototype
using namespace std;
int main(int argc, char** argv) {
    Student Erin(19,'F');
    ShowPersonInfo(Erin);//Person Info:1
    return 0;
}
void ShowPersonInfo(Person& person){//Polymorphism. Takes type Person an calls correct PersonInf
    cout << "Person Info:" << person.PersonInfo()<<endl;
}
```

Structs

```
//Struct are used to model new data types, while classes are more often used for complex real wo
#include <iostream>
using namespace std;
struct Person{
    int age; char gender;
    Person(int a=18, char g='M'){
        age = a;
        gender = g;
    }
    double PersonInfo() {
        return 0.0;
    }
private:
    int cash;
};
struct Student: Person{//Inherits Person
    Student(int a, char g){
        age = a;
        gender = g;
    }
    double PersonInfo() {
        return 1.0;
    }
};
int main(int argc, char** argv) {
    Person John(18,'M');
    Student Erin(19,'F');
    cout << "Person Info:" << John.PersonInfo()<<endl;
    cout << "Person Info:" << Erin.PersonInfo()<<endl;
    Student James{20,'M'};//Aggregate Syntax
    Person Mikayla(19,'F');
    return 0;
}
```

Operator Overloading

```
class CartesianVector{
public:
   double x, y, z;
   string cartesianString;
   CartesianVector(){
      x=1,y=1,z=1;
   }
   CartesianVector(double x1, double y1, double z1){
      x=x1,y=y1,z=z1;
   }
   CartesianVector& operator ++ (){// Uniary ++ Operator Overload
      x++;
      y++;
      z++;
      return *this;//Dereference Operator
   }
   operator const char*(){// Stream Loaded With Characters->Represent as String Object
      ostringstream boxStream;
      boxStream << "Vector("<<x<<","<<y<<","<<z<<")"<<endl;
      cartesianString  = boxStream.str();
      return cartesianString.c_str();
   }
```

```
   CartesianVector operator +(const CartesianVector& cv2){// + Operator Overload
      CartesianVector cvSum;
      cvSum.x=x+cv2.x;
      cvSum.y=x+cv2.y;
      cvSum.z=x+cv2.z;
      return cvSum;//Dereference Operator
   }
   bool operator == (const CartesianVector& cv2){
      return (x==cv2.x)&&(y==cv2.y)&&(z==cv2.z);
   }

};
int main(int argc, char** argv) {
   CartesianVector vec1(10,10,10), vec2(10,20,30);
   cout << ++vec1; //Vector(11,11,11)
   cout<<vec1+vec2;//Vector(21,31,41)
   cout<<(vec1==vec2);//0
}
```

Lambda Operation

```
// Lambda allows for one line functions. Very useful for performing any list operations in a one
vector<int> GenerateRandomVect(int vecLength, int min, int max);
void printV(vector<int> vecInt);
int main(int argc, char** argv) {
   vector<int> vInt = GenerateRandomVect(10,1,15);
   printV(vInt);///(Unsorted)
   sort(vInt.begin(),vInt.end(),[](int x, int y){return x < y;});//Lambda Function: [](int x, in
   printV(vInt);///(Sorted)
   vector<int> vEven;
   copy_if(vInt.begin(),vInt.end(),back_inserter(vEven), [](int x){return (x%2)==0;});//Lambda F
   printV(vEven);//(Even)
   int total=0;
   for_each(vInt.begin(),vInt.end(),[&](int x){total+=x;});//Lambda Function: [&](int x){total+=
   cout << "Sum:" << total;//Sum of vInt
}
void printV(vector<int> vecInt){
   cout << "(";
   for (int val: vecInt)
      cout << val << ",";
   cout<<")"<<endl;
}
vector<int> GenerateRandomVect(int vecLength, int min, int max){
   vector<int> vecValues;
   srand(time(NULL));
   int i = 0, randVal = 0;
   while (i<vecLength){
      randVal = min + rand() % ((max+1)-min);
      vecValues.push_back(randVal);
      i++;
   }
   return vecValues;
}
```

File IO

```
ofstream writeToFile;//Write
ifstream readFromFile;//Read
string txtToWrite = "", txtFromFile = "";
writeToFile.open("E:\\C Testing\\helloworld.txt",ios_base::out| ios_base::trunc);//ios_base::out
if (writeToFile.is_open()){
   writeToFile<<"Hello World"<<endl;
   writeToFile.close();
}
readFromFile.open("helloworld.txt",ios_base::in);//ios_base::in->Open For Reading
if (readFromFile.is_open()){
   while(readFromFile.good()){
      getline(readFromFile, txtFromFile);
      cout<<txtFromFile<<endl;
   }
   readFromFile.close();
}
```

Macros

```
//Macro's simply replace what's the defined name is with the value on compile
#define PI 3.14159 //Macro Constant
#define AREA_CIRCLE(r)(PI*pow(r,2))//Macro Function
```

Templates Function

```
template <typename T>
T MaxV(T v1, T v2){//Function Template That Except One Parameter
   return (v1>v2)?v1:v2;
}
int main(int argc, char** argv) {
   cout<<MaxV(1,0)<<endl;//Running Function Template With Int
   cout<<MaxV(2.4,5.2)<<endl;//Running Function Template With Int
   cout<<MaxV('A','B')<<endl;//Running Function Template With Char
   cout<<MaxV("John","Adam")<<endl;//Running Function Template With String
   return 0;
}
```

Template Class

```
template <typename T, typename U>
class Person{
   public:
      T height;
      U weight;
      Person(T h, U w){
         height = h; weight = w;
      }
      void Info(){
         cout<<"Height:" <<height<<", Weight:"<<weight<<endl;
      }
};
int main(int argc, char** argv) {
   Person<double, int> johnAdams(5.58, 182);
   return 0;
}
```

Deque

```
deque<int> nums = {2,3};//Double Ended Que
nums.push_front(1);
nums.push_back(4);
for (int a: nums){
  cout<<a<<",";//1,2,3,4,
}
```

Iterators

```
vector<int> num={0,1,2,3};
vector<int>::iterator itr;//Iterators used to point at container memory location.
for (itr = num.begin();itr<num.end();itr++){
  cout << *itr<<endl;//0,1,2,3
}
itr = num.begin();
advance(itr, 2);
cout<<*itr;//2
```

Quick Summary

Every C++ Classes are typically organized into a Header File and Implementation File. The header file contains all the function and class prototype information, while the implementation file is the meat of the class where all the functions are functions and variables are actually defined. Access to variables and function can be Public, Protected, or Private. Public is externally facing, private is internal to the class only, while protected is internal to the class and all classes that inherit the class. The virtual keyword should be used for all function that you plan to override, the destructor, or anywhere you want to use polymorphism. You can have many constructors and a lot of the time the destructor will be default. All variables shared for any object of that class type or inherited class should be defined as a static value and only static methods can access static values. Use abstract when defining a class for the sole purpose of being inherited and the & keyword is placed after the type in the argument to a method that will take advantage of polymorphism. Polymorphism allows a function to take a base class as an input argument and call functions on the base class such that if a child inherits and overrides that function, it will call the appropriate function. Structs are models more often used for data types, while classes are models more often used for real word objects. Unlike classes, everything defaults to private access in a struct. Operator overloading allows you to define functions that occur when a certain operator is applied on the class object. Lambda functions are extremely useful for creating anonymous one line functions. Macros are useful for defining any text

you want to be replaced on compile time, it can be a simple constants or functions. Templates allow for functions or classes to accept multiple types without explicitly defining each type within the function or class.

📌 **Template:** "Template" is essential for easily accepting multiple types without creating a separate function or constructor for a class.