Java to C++

1 BASICS

	C++	Java
Standard library	using namespace std; // Allow all standard library items to be used	-
Main	int main() int main(int argc, char* argv[]) int main(int argc, char** argv)	public static void main(String[] args)
Print	#include <iostream> cout << “Hello” << endl << “world” << endl; // endl == /n	System.out.println(“Hello\n” + “world”);

Scan	int year;
	string month;
	cin >> year >> month;
	cout << year << month << endl;
		int year;
		String month;
		Scanner sc = new Scanner(System.in);
		year = sc.nextInt();
		month = sc.next();
		System.out.println(year + month);

Comment	//hi OR /* hi */	same

Default function parameters	void display(int a, int b = 10, int c = 2) {
	....
	}
	
	int main() {
	display(20);
	}
	// a = 20, b = 10, c = 2
	** Set default values from the right
		-

Inline	inline void display(int a) {
	....
	}
	
	int main() {
	display(20);
	}
	// inline replaces function body into where the function is called
	** Only use for short functions
		-

Function prototype
 	
**Function is placed before calling
	void display(int a) {
	....
	}
	
	int main() {
	display(20);
	}
		-

Function prototype
 	
**Function is placed after calling
	void display(int a);
	// Declare the function before function is called
	
	int main() {
	  display(20);
	}
	
	void display(int a) {
	  ....
	}
		-

Function constant	// Function that is const can only examine object but cannot modify int hello() const;	

2 VARIABLES

	C++	Java
Global	//variable that is defined outside any functions	-
Constant	const int EXIT = 0;	private static final int EXIT = 0;
Char	char ch = 'a';	-

String	#include<string>
	string line1 = "hello";
	string line2 = "world";
	string full = line1 + " " + line2; //Awesome :)
	
	if(line1 == "hello") return true;
	//Return true
		String line1 = "hello";
		String line2 = "world";
		String full = line1 + " " + line2;
		
		if(line1.equals("hello")) return true;
		//Return true

Others	int, float, double	same

Sizeof	//Compute the bytes of datatype
	sizeof(int) //Output: 4 bytes in memory
	sizeof(int short) //Output: 2 bytes in memory
	sizeof(char) //Output: 1 byte in memory
	sizeof(float) //Output: 4 bytes in memory
		-

New/Delete	int *ptr;
	ptr = new int; //allocate memory dynamically
	*ptr = 24; //pointer pointing to address storing '24'
	
	cout << *ptr;
	
	delete ptr; // deallocate the memory so that other programs can use the memory
		-

3 POINTERS

	C++

Pointer	int age = 20;
	int *ageptr = &age;
	cout << ageptr << *ageptr;
	//ageptr gets the address of age, *ageptr gets value of age

Pass by reference	int main() {
	  int age = 20;
	  display(&age);
	  cout << age << endl;
	  //Output:10
	}
	
	void display(int *ptr) {
	  ...
	  //if we change the value of *ptr here, the value of age in main function will also be changed!
	  **Pass by reference, not by value
	
	  *ptr = 10;
	  //age has been changed to 10!
	}

Array	int main() {
	  int age[] = {1,2,3};
	  cout << *age << endl;
	  //age array contains the address of first element, hence, pointer points to the first value of array
	  //Output: 1
	}

Array subsequent elements	int main() {
	  int age[] = {1,2,3};
	  cout << *(age+2) << endl;
	  //Output: age[2]
	}

Array pointers	int main() {
	  int arr[] = {1,2,3,4,5};
	  display(*arr, *(arr+5));
	}
	
	void display(const int * start, const int *end) {
	  const int *ptr;
	  for(ptr = start; ptr != end; ptr++)
	  cout << *ptr << " ";
	  //Output: 1 2 3 4 5
	}

Others	int a = b is setting a's VALUE to b's VALUE int* a = &b is setting a's VALUE to the ADDRESS of b int& a = b is setting a's ADDRESS to b's ADDRESS (a is a reference to b)

4 STRUCTURES

	C++	
	Dot operator	**Arrow operator**

Struct	struct student {
	  int id;
	  char gender;
	} lame, lol; //names of student
	
	int main() {
	  student lame, lol = {123, 'M'};
	  lame.id = 1234; //lame's id has been changed to 1234
	}
		struct student {
		  int id;
		  char gender;
		};
		
		int main() {
		  student lol;
		  student *lolptr;
		
		  lol.id = 1234;
		  lolptr = &lol;
		  lolptr -> gender = 'M';
		
		  cout << lolptr -> id >> endl; //print lol's id
		  cout << lol.gender << endl; //print lol's gender
		}

Functions pass by value or reference	void value(student s) {
	  cout << s.id << endl; //Pass by value
	}
		void reference(student *s) {
		  cout << s -> id << endl; //Pass by reference
		}

Nested structure	struct address {
	  int house_no;
	  string street_name;
	};
	
	struct student {
	  string name;
	  address add;
	};
	
	int main() {
	  student lol;
	  lol.name = "lol";
	  lol.add.house_no = 123;
	}
		struct address {
		  int house_no;
		  string street_name;
		};
		
		struct student {
		  string name;
		  address add;
		};
		
		int main() {
		  student lol;
		  student *lolptr = &lol;
		  lol -> name = "lol";
		  lol -> add.house_no = 123;
		}

Union	//variables share the same values
	union employee{
	  int id;
	  float salary;
	}
	int main() {
	  employee lol;
	  lol.id = 25;
	  cout << salary;
	  //Output: 25, since id = 25, salary = 25 as well
	}	

5 OPERATORS

	C++	Java
Ternary	age >= 15 ? cout << “Older than 14” : cout << “Younger than 15”	same
Address	int age; cout << &age << endl; //memory location of age	-

6 OVERLOADING OPERATORS

	C++

Overloading operator ‘+’ ‘-’	class Marks {
	int a;
	int b;
	
	public:
	Marks(int ia, int ib) {
	int a = ia;
	int b = ib;
	}
	
	void display() {
	cout << a << endl << b << endl;
	}
	
	Marks operator + (Marks m) {
	Marks temp;
	temp.a = a + m.a;
	temp.b = b + m.b;
	}
	
	Marks operator - (Marks m);
	}
	
	Marks Marks :: operator - (Marks m) {
	Marks temp;
	temp.a = a - m.a;
	temp.b = b - m.b;
	}
	
	int main(){
	Marks mark1(10,20), mark(30,40);
	Marks mark3 = mark1 + mark2;
	Marks mark4 = mark2 - mark1;
	mark3.display(); //Output: 40, 60
	mark4.display(); //Output: 20,20
	}

Array	class Marks {
	  int subjects[3];
	
	  public:
	  Marks(int sub1, int sub2, int sub3) {
	  subjects[0] = sub1;
	  subjects[1] = sub2;
	  subjects[2] = sub3;
	  }
	
	  int operator[] (int position) {
	  return subjects[position];
	  }
	}
	
	int main(){
	  Marks lame(1,2,3);
	  cout << lame[0] << end1 << lame[1] << endl << lame[2] <<endl;
	}

Function call	class Marks {
	  int mark;
	
	  public:
	  Marks(int m) {
	  mark = m;
	  }
	
	  void display {
	  cout << mark << endl;
	  }
	
	  Mark operator() (int mk) {
	  mark = mk;
	  return *this;
	  }
	}
	
	int main(){
	  Marks lame(5);
	  lame.display();
	  lame(44);
	}

7 ERROR HANDLING

	C++

Assert	#include <cassert>
	assert(month <= 12);

Exception handling	#include <exception>
	
	int main() {
	  int a = 10, b = 0;
	  int c;
	
	  try {
	  if(b==0) {
	  throw “b boooooooo”;
	  }
	  c = a/b;
	  } catch (const char *e) {
	  cout << e.what() << endl;
	  }
	}

Exception handling e.g. Runtime error	#include <exception>
	
	int main() {
	  int a = 10, b = 0;
	  int c;
	
	  try {
	  if(b==0) {
	  throw runtime_error “divide by 0 error”;
	  }
	  c = a/b;
	  } catch (runtime_error &error) {
	  cout << “Exception occured” << endl;
	  cout << error.what();
	  }
	}

Function exception	#include <exception>
	
	void test() throw(int, char, runtime_error) {
	  throw 20;
	}
	
	int main() {
	  try {
	  test();
	  } catch(int e) { //Since throw 20 which is integer, catch!
	  cout << “integer exception” << e << endl;
	  }
	}

User defined exception	#include <exception>
	
	class OverSpeed : public exception {
	  int speed;
	  public :
	  const char* what() {
	  return “Bad speed\n”;
	  }
	
	  void getSpeed() {
	  cout << speed << endl;
	  }
	
	  void setSpeed(int speed) {
	  this->speed = speed;
	  }
	}
	
	class Car {
	  int speed = 0;
	
	  void accelerate() {
	  for(;;) {
	  speed += 10;
	  if(speed >= 250) {
	  OverSpeed s;
	  s.setSpeed(speed);
	  throw s;
	  }
	  }
	  }
	}
	
	int main() {
	  Car car;
	
	  try {
	  car.accelarate();
	  } catch(OverSpeed s) {
	  cout << s.what() << endl;
	  s.getSpeed();
	  }
	}

7 GENERIC PROGRAMMING

	C++

Template	//Instead of writing 2 similar body functions
	int max(int x, int y) { return (x > y); }
	int max(float x, float y) { return (x > y); }
	
	//Use generic:
	
	template <typename T> void display(T x, Ty) {
	  cout << x << endl << y << endl;
	}
	
	template <typename T1, typemame T2> void hello(T1 x, T2y) {
	  cout << x << endl << y << endl;
	}
	
	int main() {
	  display(100, 200);
	  display("HAHA", 26);
	  hello(3.3, "LAME");
	  return 0;
	}

C++ OBJECT ORIENTED PROGRAMMING

1 CONSTRUCTOR

Constructor

```
class Student {
  private:
      string name;
      int age;

  public:
     Student() {
          name = “lame”;
          age = 100;
     }

     Student(string iname, int iage = 100) {
          name = iname;
          age = iage;
     }

     Student(string iname, int iage) {
         name = iname;
          age = iage;
     }

      void display() {
         cout << getName() << endl;
      }
};

int main() {
   Student lame; //Calls default constructor Student()
   lame.display();

   Student lol(“haha”);  //Student(string name, int age = 100)
   haha.display();

   Student lol(“lol”, 100); //Student(string name, int age)
   lol.display();
   return 0;
}
```

2 DESTRUCTOR

Destructor

```
class Student {
  private:
      string *name;
      int *age;
```

	public:
	Student(string iname, int iage) {
	name = new string;
	age = new age;
	*name = iname;
	*age = iage;
	}
	
	~Student() {
	delete name;
	delete age;
	cout << “all memories are released” << endl;
	}
	
	void display() { ... }
	};
	
	int main() {
	Student *lame = new Student(“lame”, 100); //Calls default constructor Student()
	lame -> display();
	delete lame; //Calls destructor ~Student()
	
	return 0;
	}

8 LOCAL CLASS

	C++

Local class	void studentList() {
	class Student {
	public:
	string name;
	int age;
	
	void display() { ... };
	};
	
	Student lame;
	lame.name = “lame”;
	lame.age = 20;
	lame.display();
	}
	
	int main() {
	studentList();
	return 0;
	}

3 PRIVATE CLASS

	C++

Private class	#include <string>
	
	class Student {
	  private :
	  string name;
	
	  int getName() {
	  return name;
	  }
	
	  public :
	  void display() {
	  cout << getName() << endl;
	  }
	
	  void setName(string iname) {
	  name = iname;
	  //Can access private attribute name since it is in the class
	  }
	};
	
	int main() {
	  Student lame;
	  lame.setName(“lame”);
	  lame.display();
	
	  return 0;
	}

4 PUBLIC CLASS

	C++

Public class	#include <string>
	
	class Student {
	  public :
	  string name;
	
	  void display() {
	  ....
	  }
	};
	
	int main() {
	  Student lame;
	  lame.name = “lame”;
	  lame.display();
	
	  Student *lol = new Student();
	  lol -> name = “lol”;
	  lol -> display();
	  return 0;
	}

Define methods outside class	#include <string>
	
	class Student {
	  public :
	  string name = “haha”;
	  void display();
	};
	
	void Student :: display() {
	  cout << Student :: name << endl;
	}
	
	int main() {
	  Student lame;
	  lame.name = “lame”; //Override name “haha”
	  lame.display();
	  return 0;
	}

5 STATIC

	C++

Static	class Student {
	  public :
	  static int count;
	
	  Student() { count++; }
	
	  void studentTotal() {
	  cout << count << endl;
	  }
	
	  static void studentCount() {
	  cout << count << endl;
	  }
	};
	
	int Student :: count = 0;
	
	int main() {
	  Student lame;
	  Student lol;
	  Student haha;
	  lame.studentTotal();
	  Student :: studentCount(); //Output: 3
	  cout << Student::count << endl; //Output: 3
	  return 0;
	}

6 FRIEND FUNCTION

	C++

Friend function/class	class Student {
	  public :
	  string name;
	  int age;
	
	  Student(string iname, int iage) {
	  name = iname;
	  age = iage;
	  }
	
	  void studentInfo() {
	  cout << name << endl << age << endl;
	  }
	
	  friend void display(Student student);
	  friend class Teacher;
	};
	
	void display(Student student) {
	  cout << student.name << endl << student.age << endl;
	}
	
	int main() {
	  Student lame(“lame”, 100);
	  display(lame);
	  return 0;
	}

7 INHERITANCE

	C++

Inheritance	class Person { //Base class
	  protected :
	  string name;
	  public :
	  int age;
	
	  void setName(string iname) {
	  name = iname;
	  }
	
	  void setAge(int age) {
	  age = iage;
	  }
	};
	
	class Student : public Person { //Subclass
	  public :
	  int id;
	
	  void setId(string iid) {
	  id = iid;
	  }
	
	  void introduce() {
	  cout << name << endl << age << endl << id << endl;
	  }
	};
	
	int main() {
	  Student lame;
	  lame.setName(“lame”);
	  lame.setAge(100);
	  lame.setId(12345);
	  lame.introduce();
	  return 0;
	}

8 MULTIPLE INHERITANCE

	C++

Multiple inheritance	class Father { //Base class
	  public :
	  int height;
	
	  void askFather() {
	  cout << “Ask father” << endl;
	  }
	};
	
	class Mother { //Base class
	  public :
	  string skincolour;
	
	  void askMother() {
	  cout << “Ask mother” << endl;
	  }
	};
	
	class Child : public Father, public Mother { //Subclass
	  public :
	  void askParents() {
	  cout << “Ask parents” << endl;
	  }
	
	  void setColourAndHeight(string icolour, int iheight) {
	  height = iheight;
	  skincolour = icolor;
	  }
	
	  void display() {
	  cout << height << endl << skincolour << endl;
	  }
	
	};
	
	int main() {
	  Child lame;
	  lame.setColourAndHeight(“white”, 100);
	  lame.askParents();
	  lame.askFather();
	  lame.askMother();
	  return 0;
	}

	class Father { //Base class
	  public :
	  int height;
	
	  Father() { }
	};
	
	class Mother { //Base class
	  public :
	  string skincolour;
	
	  Mother() { }
	};
	
	class Child : public Father, public Mother { //Subclass
	  public :
	  Child(int x, string colour) : Father(), Mother() {
	  height = x;
	  skincolour = colour;
	  }
	
	  void display() {
	  cout << height << endl << skincolour << endl;
	  }
	};
	
	int main() {
	  Child lame (5, “white”);
	  lame.display();
	  return 0;
	}

Access overridden methods	class Person {
	  public:
	  void hello() { … }
	}
	
	class Student : public Person {
	  public:
	  void hello() { Person :: introduce(); } //Access overriden method
	}
	
	int main() {
	  Student lame;
	  lame.hello();
	  lame.Person :: hello(); //Access overriden method
	}

9 POLYMORPHISM

	C++

Virtual	class Person { //Base class
	  public :
	  virtual void hello() {
	  cout << “Person” << endl;
	  }
	};
	
	class Student : public Person { //Base class
	  public :
	  void hello() {
	  cout << “Student” << endl;
	  }
	};
	
	class Farmer : public Person { //Base class
	  public :
	  void hello() {
	  cout << “Farmer” << endl;
	  }
	};
	
	void who(Person &p) {
	  p.hello();
	}
	
	int main() {
	  Farmer lame;
	  Student lol;
	
	  who(lame); //Output Person if no virtual word, else Farmer
	  who(lol); //Output Person if no virtual word, else Student
	
	  return 0;
	}

10 ABSTRACT

	C++

Abstract	class Person { //Abstract class -> cannot create object for this class e.g. Person lame; << NO!
	  public :
	  virtual void hello() = 0; //We need virtual keyword so that subclass's hello() is called instead of base class
	};
	
	void Person :: introduce() {
	  cout << "Person" << endl;
	}
	
	class Student : public Person { //Base class
	  public :
	  void hello() {
	  cout << "Student" << endl;
	  Person :: introduce();
	  }
	};
	
	int main() {
	
	  Student lol;
	  lol.hello();
	
	  return 0;
	}

Diamond	class Animal {
	  public :
	  Animal() { }
	  virtual void hello() = 0;
	};
	
	class Tiger : virtual public Animal {
	  public : Tiger() { }
	};
	
	class Lion : virtual public Animal {
	  public : Lion() { }
	};
	
	class Liger : public Tiger, public Lion {
	  public : Liger() { }
	};
	
	int main() {
	  Liger lame; // Animal > Tiger > Lion > Liger
	  lame.hello();
	
	  return 0;
	}

11 MULTIPLE CLASSES

	C++

Person.h	#ifdef PERSON_H
	#define PERSON_H
	
	class Person {
	  public :
	  Person();
	  void hello();
	};
	
	#endif

Person.cpp	#include “Person.h”
	
	Person :: Person() {
	  cout << “Person constructor” << endl;
	};
	
	void Person :: hello() {
	  cout << “Hello person” << endl;
	}

main.cpp	int main() {
	  Person lame;
	  lame.hello();
	
	  return 0;
	}

C++ FILE HANDLING

Create and open file	#include <iostream>
	#include <fstream>
	
	int main() {
	  fstream file;
	  file.open("filename.txt", ios :: in | ios :: out | ios :: trunc); // trunc creates file if not exist, recreates file if exists :(
	
	  if(file.is_open()) {
	  cout << "Yeah!" << endl;
	  }
	
	  file.close();
	  return 0;
	}

Write file	#include <iostream>
	#include <fstream>
	
	int main() {
	  ofstream file("filename.txt");
	
	  if(!file.is_open()) {
	  cout << "Unable to open file" << endl;
	  } else {
	  file << "WRITE CONTENT IN!!" << endl;
	  file.close();
	  }
	
	  return 0;
	}

Read file	#include <iostream>
	#include <fstream>
	
	int main() {
	  ifstream file;
	  file.open("filename.txt");
	
	  if(!file.is_open()) {
	  cout << "Unable to open file" << endl;
	  } else {
	  string line;
	  while(file.good()) {
	  getline(file, line);
	  }
	  file.close();
	  }
	
	  return 0;
	}

Append file	#include <iostream>
	#include <fstream>
	
	int main() {
	  fstream file("filename.txt", ios :: in | ios :: out | ios :: app); // app creates file if not exist, does not recreates if exists :)
	
	  if(!file.is_open()) {
	  cout << "Unable to open file" << endl;
	  } else {
	  file.seekg(0);
	  while(file.good()) {
	  getline(file, line);
	  }
	  file.close();
	  }
	
	  return 0;
	}

Read from nth character	#include <iostream>
	#include <fstream>
	
	int main() {
	  fstream file("filename.txt", ios :: in)
	
	  if(!file.is_open()) {
	  cout << "Unable to open file" << endl;
	  } else {
	  cout << file.tellg() << endl; // tellg counts number of characters in file
	  string line;
	  file.seekg(2); // read from 2nd character :p
	  getline(file, line);
	  file.close();
	  }
	
	  return 0;
	}

Write from nth character
 	
*file.seekp(5,) can also be used to offset from beginning, end position
	#include <iostream>
	#include <fstream>
	
	int main() {
	  fstream file("filename.txt", ios :: out)
	
	  if(!file.is_open()) {
	  cout << "Unable to open file" << endl;
	  } else {
	  cout << file.tellp() << endl; // tellp counts number of characters in file (Currently 0 if no content)
	  file << "I rocks" << endl;
	  file.seekp(5); // write until 5th character :p (Output: I roc)
	  file.close();
	  }
	  return 0;
	}