C++

[https://www.techiedelight.com/difference-between-static-dynamic-binding-cpp/#:~:text=Static%20binding%20happens%20when%20all,function%20call%20at%20compile%2Dtime](https://www.techiedelight.com/difference-between-static-dynamic-binding-cpp/" \l ":~:text=Static binding happens when all,function call at compile-time).

<https://www.geeksforgeeks.org/polymorphism-in-c/>

Terminology:

1. Abstractions – classes
2. Procedure – Functions
3. Encapsulation- using classes
4. Polymorphism- same name of function, but diff body

Identifiers: name of variable

* Should not start with numbers.
* Should use letters, numbers, “\_” only.
* Space is not allowed
* Should not use preserved words like **int**. but we can use **last\_int**

C doesn’t support OOPS Concept fully, because it doesn’t have private members, But C++ supports.

ASCII codes of **A**=65 **a**=97

|  |  |
| --- | --- |
| 1)int | 32 bits (1 for sign,23 for significant,8 for exponent) |
| 2)unsigned int | 32 bits |
| 3)char | 8 bits (for small numbers and characters) |
| 4)float | 32 bits (real numbers) |
| 5)double | 64 bits |
| 6)long int | 64 bits |
| 7)long double | 96 bits |

**const long double N = 6.02 \* 10^23;**

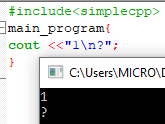
* Multiplication & division have same precedence. Addition & Subtraction have same precedence.1st precedence is more than 2nd one. Operators of same precedence will be evaluated from left to right.
* Gaps can be there in between the operators.

**int a; int &a =b;** //in the function, if u change a that will change b, and if u change b that will change a.

**∙** In C++, 2/3 = 0, 2.0/3 = 2/3.0 = 2.0/3.0 = 0.66666…

**∙** Avogadro + 10 = Avogadro, precision is up to 7 digits only.

We have to use **cout << fixed; cout.precision(2);** before every cout.



**∙**10 **%** 7 = 3 (**remainder function**)

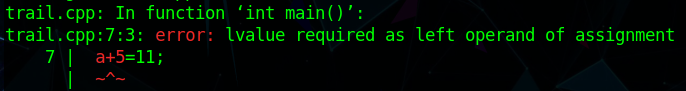
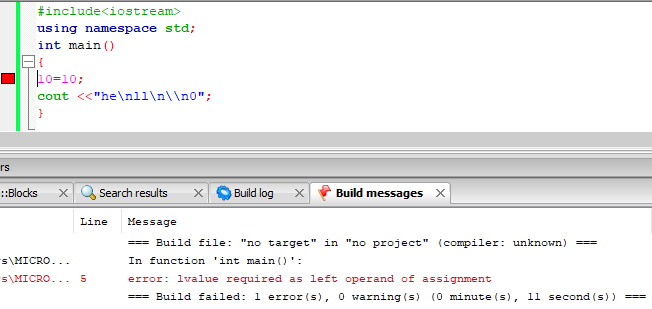
**∙max (a, b); min (a, b);** are defined function’s in c++.

**∙abs (-200)** = 200

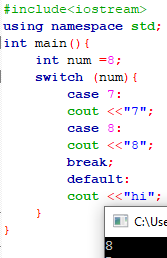
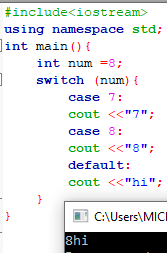
**∙**power function (**^**) **pow(a,b)** = a^b //but be careful with this, read extra reading material.

**∙**s++ and ++s will give diff results. //read extra reading material

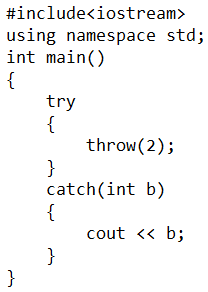
Lvalue operand error:



Switch:

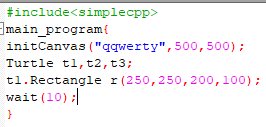
Throw, Try, Catch:



**sqrt(24)** is square root function.

TURTLE SIMULATOR

**initCanvas(“canvas”, 500, 500);** is similar to **turtleSim();** but u have to create one turtle with **Turtle n1,n2;** in canvas

 this has no meaning.

in canvas, if we use turtle, then it generally starts from midpoint point of canvas

x axis rightward, y-axis downward

**Circle c (100,100,25);** first two are coordinates of centre, next one is radius.

**Rectangle r(100,100,30,25);** first two are coordinates of centre, next two are width and height respectively

**Line l(100,100,200,200);** first two are coordinates of first point and next two are coordinates of second point.

**Text t(100,100,”text is being written”);**  first two are coordinates of centre of text

**Rectangle r(100,100,textWidth(“text is being written”),textHeight());**

**name.moveTo(250,250);** to move any shape to new centre.

**name.move(5,5);** to move any shape by 5 units right and below.

**name.scale(5);** that shape will be scaled 5 times with centre at same position.

**name.rotate(2);** that shape will rotate **right** about centre by angle in **radians**.

**cout << sine(90);** //will print 1

**cout << sin(1.57);** //will print 1

Scale and rotate doesn’t apply on text.

**name.setColor(COLOR(255,255,0));** or **name.setColor(COLOR(“red”));** to change the color of the border, primary colors are red, green, blue.

**r.setFill(true);**

**r.setColor(COLOR(“red”));** to change the color inside the shape.

**float x=r.getX(),y=r.getY(),z=r.getOrientation(),W=r.getScale();**

to get x coordinate of centre, y coordinate of centre, angle rotated with respect to centre(in radians), scaling factor.

**r.imprint();**  will print that shape on the canvas and it can’t be moved.

**float z=r.getClick();** number 2^16\*x + y will be stored in z,(x,y) are coordinates of click position.

**getClick();** also used to wait the program until we click on the window

**If (2 <= 5) {cout << “program is correct”;};**

**X < Y** less than **X > Y** greater than **X >= Y** greater than or equal to

**X <= Y** less than or equal to **X == Y**(don’t forget this)equal to **X != Y** not equal to

**Condition1 && condition2** implies intersection.

**Condition1 || condition2** implies union.

**!(condition)** implies negation.

Don’t write as **0 <= x <= 10**,we have to write as **0 <= x && x <= 10**

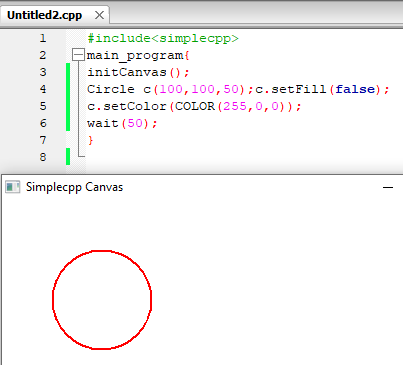
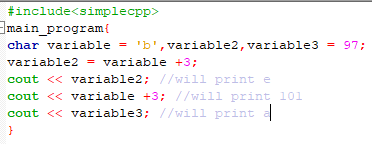
**!** has higher precedence than **&&** which has higher precedence than **||**

--> 3 ways of debugging

1)put print statements to see whether intermediate values are correct or not.

2)manually trace the problem execution.

3)symbolically trace the problem.(didn’t understand this)



WHILE LOOP

**while(true){**

**cin >> nextmark ;**

**if(nextmark <0) {break;};**

**if(nextmark > 100) {continue;};** because marks are out of 100.continue tell that to ignore that iteration and move to next iteration.

**sum = sum + nextmark;**

**count++ ;**

**};**

if we assign some variable inside while(…),then it is treated as true always.

if we use **continue** in some loop, then remaining body is ignored and then it will run next iteration.

If **break** is used inside a while statement which is itself nested inside another for while, then only inner while statement is terminated.

FOR LOOP

**for(initialization; condition; update) {body}** //can be without curved brackets

New variable can be introduced in the initialization, but those are meaningless outside the ‘for’ loop.

FUNCTIONS

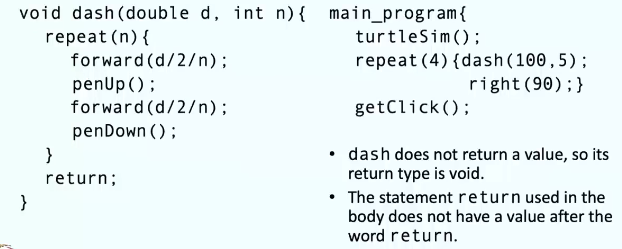
**returned\_variable-type name\_of\_function(variabletype1 variable1name,variabletype2 variable2name) {main body;**

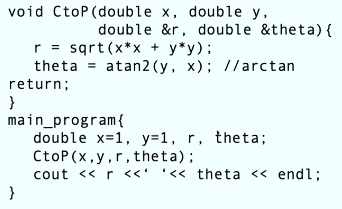
**return somevariablename;}** // we should use only **return;** for void type functions

We should use **void** in place of returned\_variable-type for non-returning functions.

We should not use **cout << name\_of\_function;** // for void type functions

If we use & before any argument name in a function, then that variable is passed by reference.





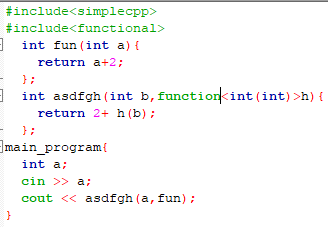
Passing a function to another function:

**#include<functional>** //we have to include this directory

Beside arguments we have to write

**function<return\_type(argument\_type)> name\_of\_function**

Here name of function need not be same as in definition of the passed function, this name should be used in the body of the function to call the passed function.

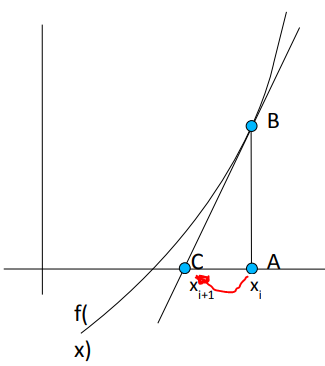




Taylor series can be used even if x-x0 >1.

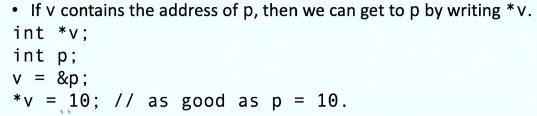
 Bisection method for calculating roots

 newton-raphison method, it uses the differentiation.



POINTERS

**double \*variable;** can be interpreted as variable is address of some double variable, which is represented as hexadecimal number.



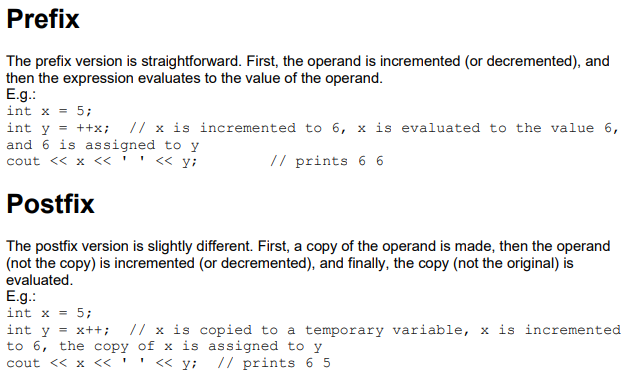
**delete name\_of\_pointer;**//can be used to delete pointer

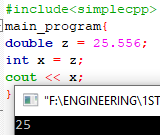
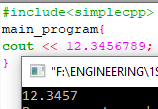
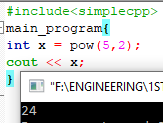
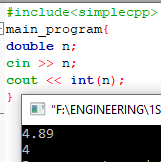
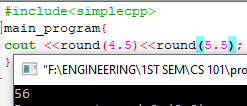
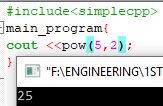
**Int\* ptr;**

**ptr = NULL;**

**cout << ptr->member;** //gives error

you may check if a pointer doesn’t point to anything by writing **ptr == NULL**



**round(2.34);**returns 2,it roundes off to nearest integer

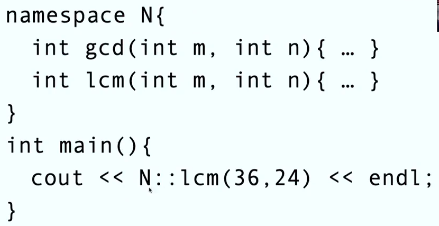
NAMESPACE

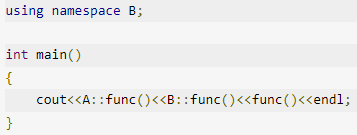
If u don’t use **#include<simplecpp>** u have to use

**#include<iostream>**

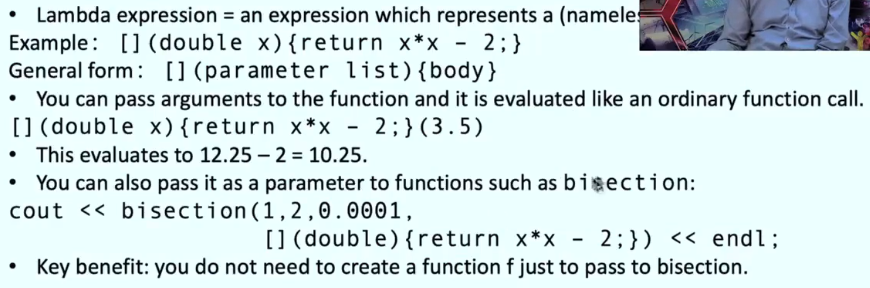
**using namespace std;**

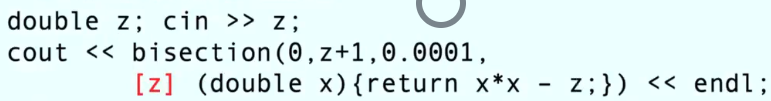
if u don’t use simplecpp, u can’t do graphics.

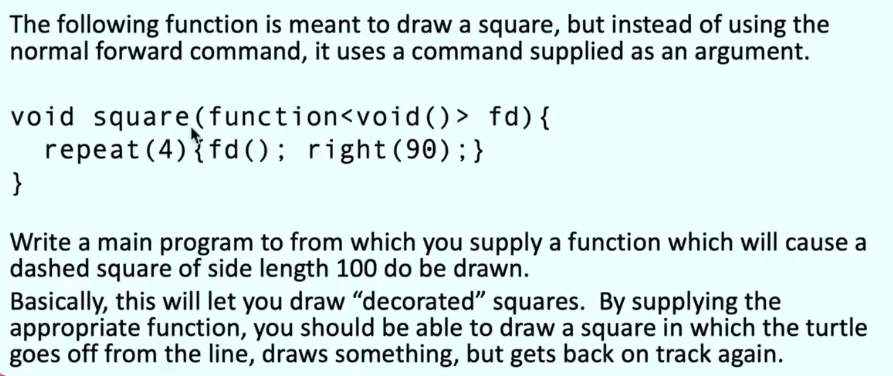


we can use like this.

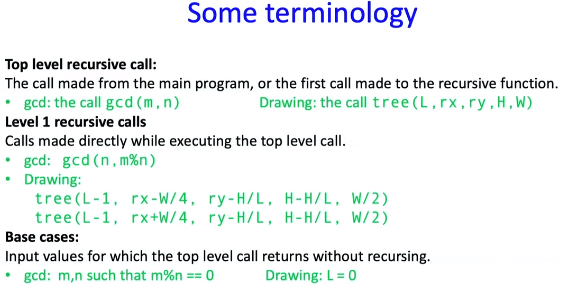
Lambda Expressions:







RECURSIVE FUNCTIONS

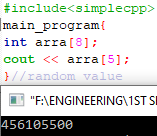


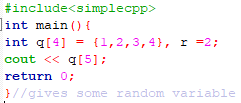
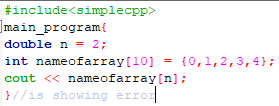
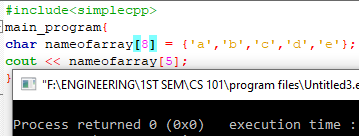
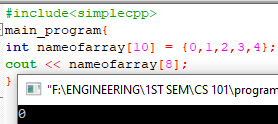


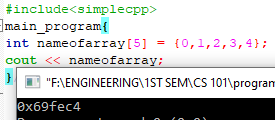
Recursive function is very similar to normal function, but in the body of the program we use the same function with lower order. **Be careful at Base cases**.

ARRAYS

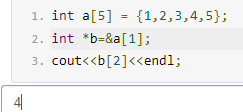
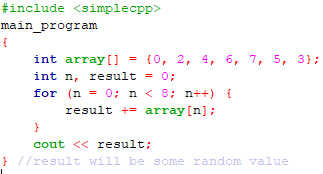
Arrays name is an address data type actually, and it is address of first element.

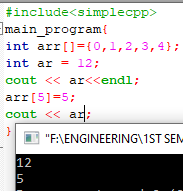






Address of first element will be printed.

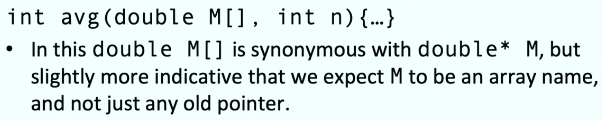


it will erase the elements if u increase the no of elements after defining. **Not so sure**

**Turtle random\_name[i];**

**Circle random\_name[i];**

**random\_name[i].reset(x,y,r);** //arrays for graphical objects



**int \*a1;**

**a1 = new int [5];** //an array of length 5 is created and address of first element is assigned to a1.

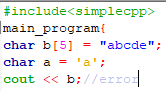
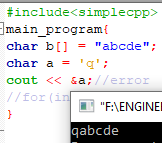
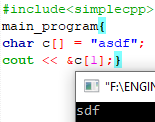
Segmentation Error:

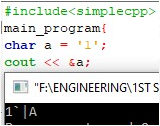
When heap is accessing more memory.

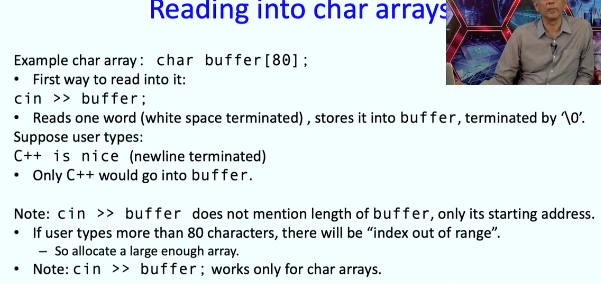
CHARACTER ARRAYS

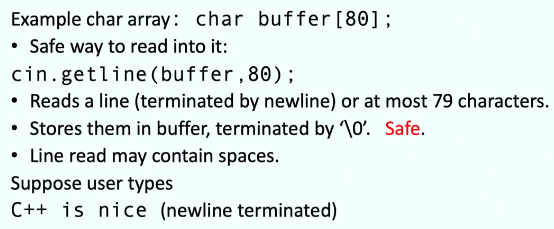
**char n1[20] = “Ajanta”, n2[] = “ellora”;**//only double quotes

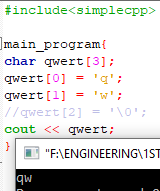
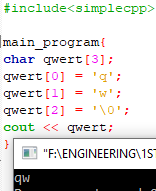
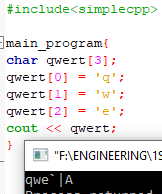
1st way to read into char array: **char char\_arr\_name[10];cin >> char\_arr\_name;** in this way, if white space come, it will stop reading into the array.Ex: if you enter **I am a student** ,it will store only **I**.

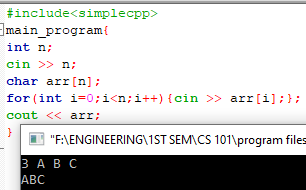
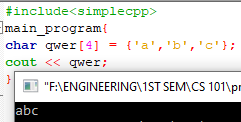
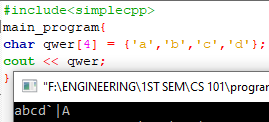
  



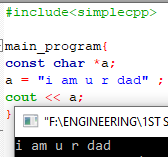








Character Constants:

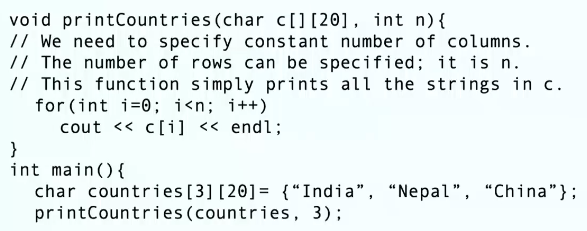


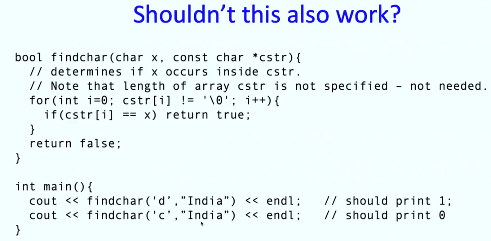
2 Dimensional Arrays:

**int pqr[2][3] = {{1,5,7},{13,6,2}};**

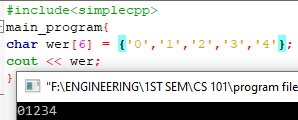
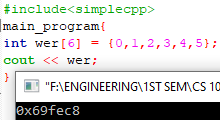
**char countries[3][20] = {“India”,”Nepal”,”China”};**

**cout <<countries [2];** //prints China





“India” or any text enclosed with double quotes is stored as a character constant. And it is stored as a address of first element.

Sorting an Array:

1) Selection Sort: It finds the largest number in the array and swaps with last element. No of comparisons is around n2/2.

2) Bubble sort: it swaps the every consecutive pairs if left one is larger than right one. It will keep on swapping the elements. If a loop got completed with 0 swaps, then the function will stop.

3) Merge sort: it will divide into 2 parts recursively, until it reaches 1/2/3 elements. Then it will merge recursively.

STRUCTURES & CLASSES

If you use class, then every member function and member will be private. If you use struct, then every member function and member will be public.

U can change members or member function by using **public:** and **private:**

**struct structure\_type{double name\_of\_member;char random\_name[5];};**//class can be used in place of struct

**structure\_type p,q;**

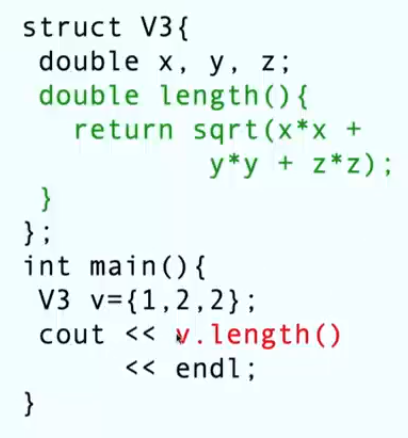
**p.name\_of\_member = 50;**

**p.random\_name = {‘a’,’b’};**

a structure\_type is a user defined data type just like int, double. struct is fixed for all structures

Member Functions:

Member functions should be public, else we can’t use them. We can’t even assign a value of a private variable to a variable in main program and We can’t even print that private variable.



Constructor:

1) default constructor:- is already there, which won’t do anything.

2) **V3() {x = 1; y=2;};** - constructor with no arguments

3) **V3(int a,int b,int c){x = a+2;y = b-1; z = a+b-c;};**- constructor with arguments

There can be many constructors for a structure. But those constructors should be different in number of arguments, if same atleast they should not have same data types of arguments in the same order.

Destructor:

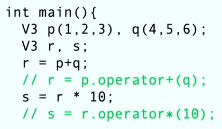
**~V3 (){cout << “destructor is called”;};** with no arguments

Destructor is called by a function, when it ends. Destructors of the objects defined in that function are called.

Operator Overload:

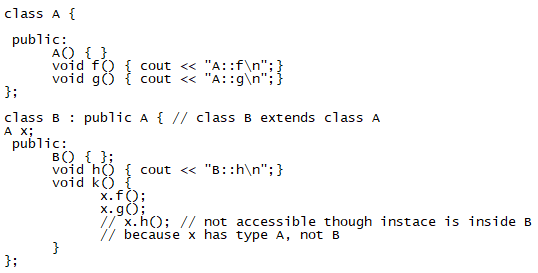
Operator’s precedence doesn’t change if we overload.

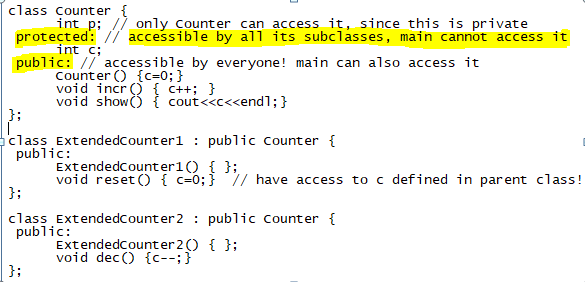
**V3 operator +(V3 random){V3 dummy; dummy.x = x + random.x; dummy.y =y + random.y;return dummy;};**

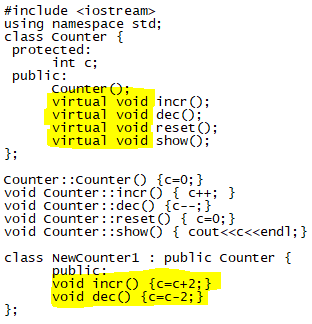


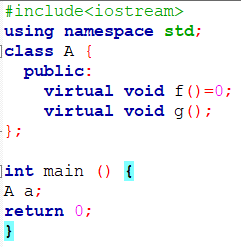
Operator **-> x -> y** is same as **(\*x).y**

INHERITANCE









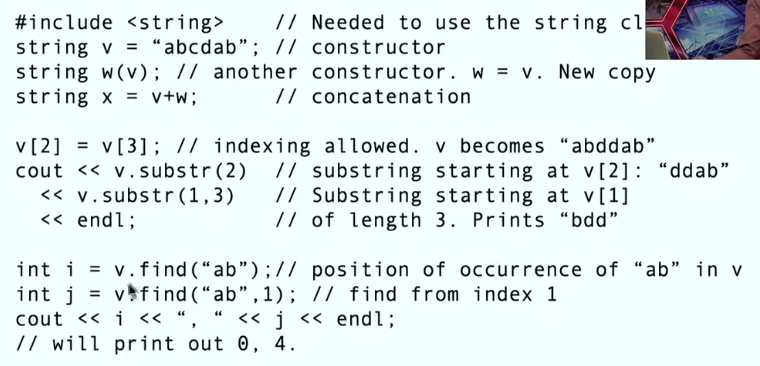
we can’t instant these type of structures or classes. (because of **=0**)

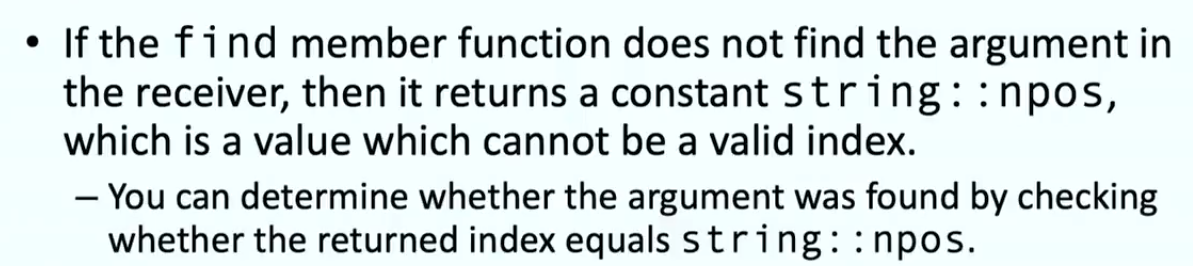
What is static?

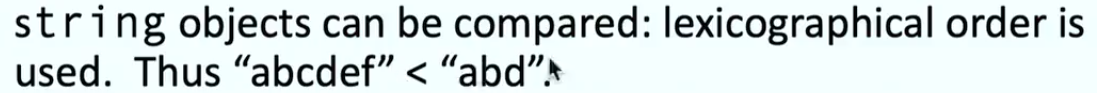
**this -> member\_name**

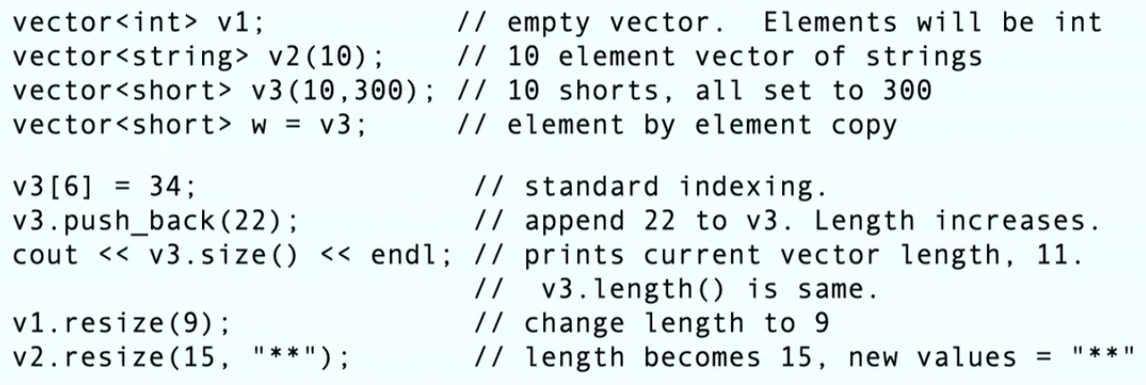
# 

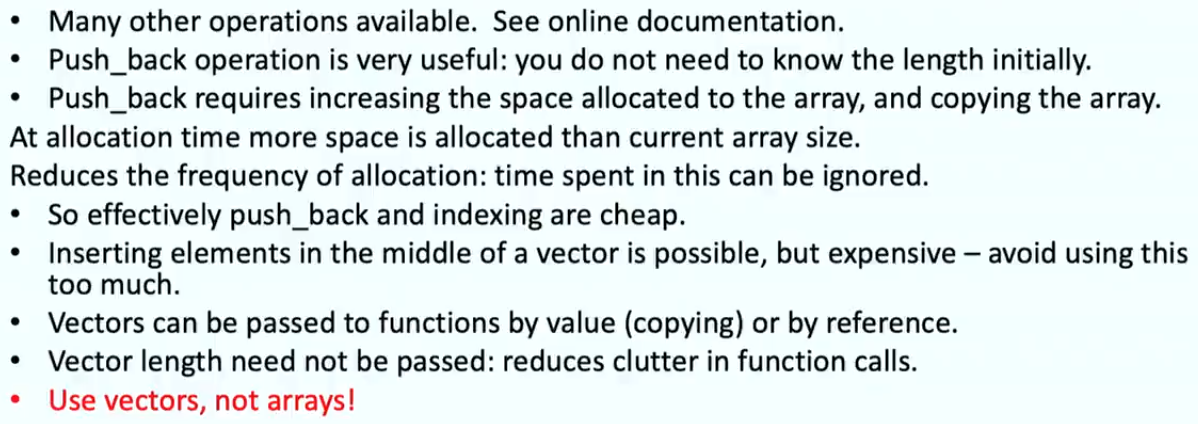
VECTORS & STRINGS



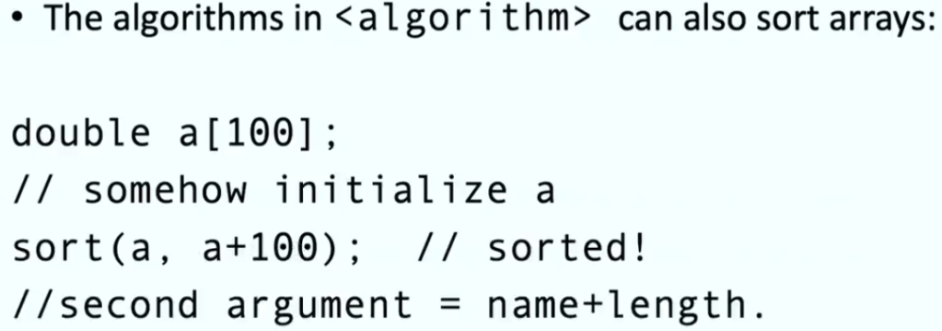




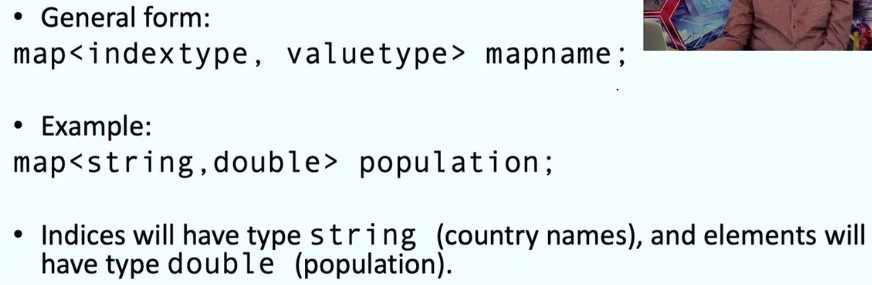


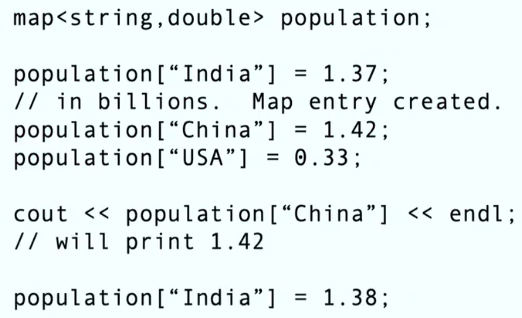


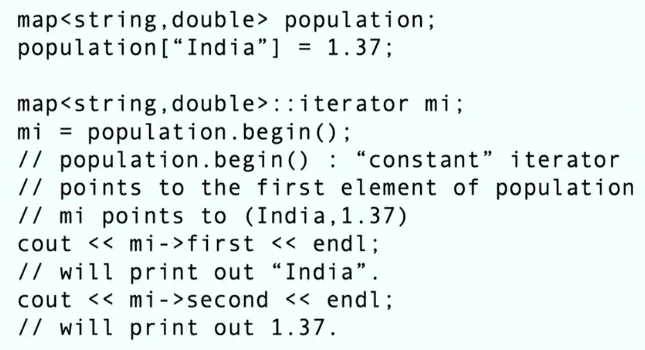




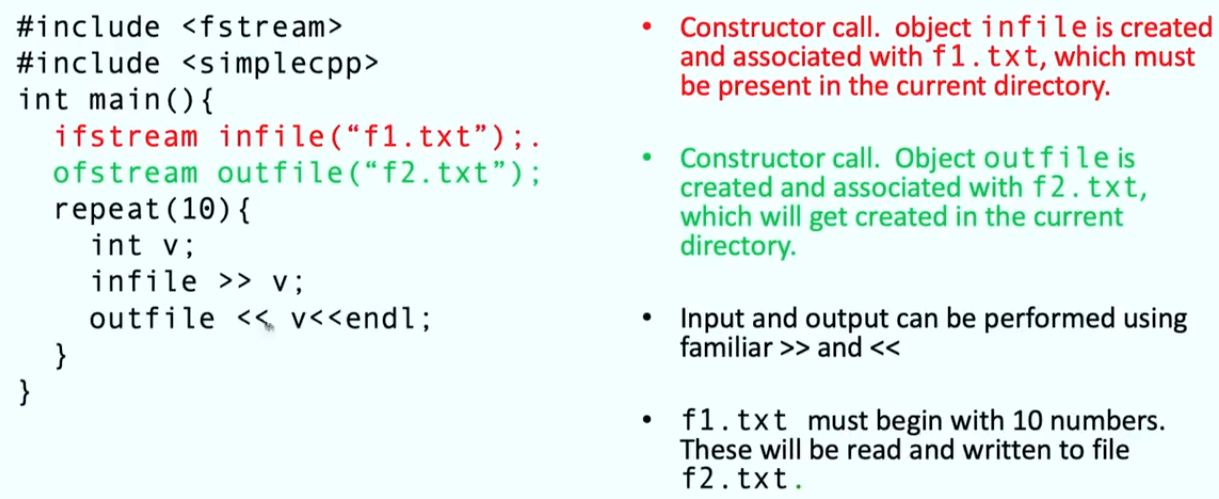
MAPS

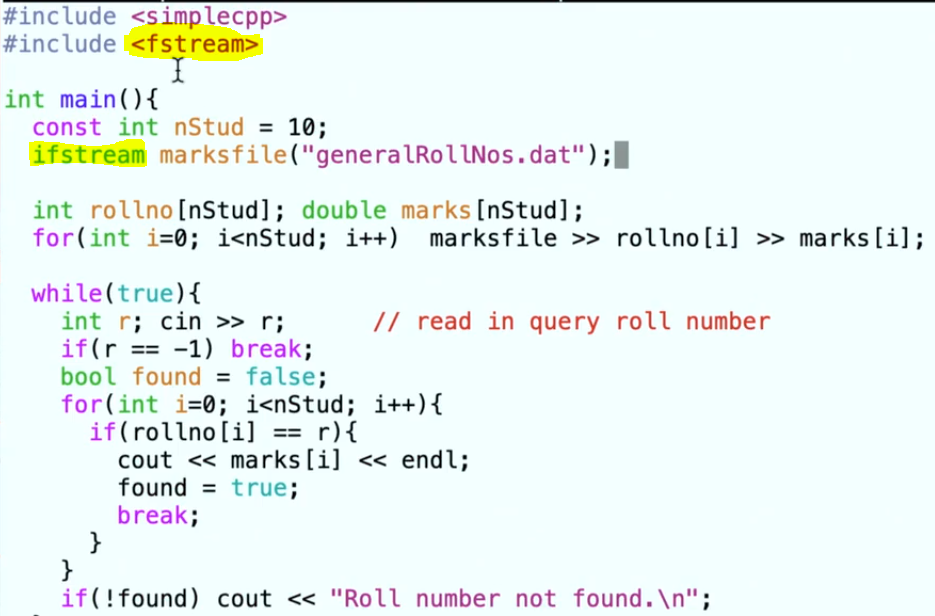






To take input from a text file (.txt)





**PENDING WORKS**

Optional videos:- 2nd lecture 3rd part, 4th lecture 1st part (3videos ,1quiz), 6th lecture 4th part(5 videos),

Option lab questions-every week

**“do while statement” read from textbook.** 

