C++ study documentation

1. sizeOf() - operator to find the memory size of a variable.

For array :

int a=19;

cout<<sizeof(a)<<endl;

int arr[10];

int arr2[20]={1,2,3,4,5}; //remaining indexes filled with 0

cout<<sizeof(arr2)<<endl; // 20\*4 = 80

For char :

char c=10;

char ch[10]="dinesh";

cout<<" char size : "<<sizeof(ch)<<endl; // 10\*1 = 10

char ch2[]="dinesh";

cout<<" char size : "<<sizeof(ch)<<endl; // 7\*1 = 7 , {'d','i','n','e','s','h','/0'}

For pointers:

For any types of pointers , sizeof(ptr) return only the size of the pointer , not the size of varibale it stores.

Ex:

int \*arr = new int[10];

cout << sizeof(arr) << endl; // 8 ,since the size of any ptr in 64 bit system is 8

Datatypes :

int,char,float,short,long,double,bool

bool val=true // cout<<val return 1 or 0

if(val) // if(1)

**Type casting:**

1 . (int)value

2. Anything to string:

To\_string();

**Scope of variables :**

1.Local

2. Global

Pass by value , Pass by reference , Pass by Pointers:

NOTE: All the data are pass by value by default , except the arr[].

int arr[10]; // Pass by value

int \*arr=new int[10]; // Pass by Pointers:

func(int &arr) // Pass by reference

string \*str=new string();

\*str="dinesh"; // here , str stores the address of the string , not the value

To access the value, we need to dereference by using ‘\*’.

void modify(vector<int> &vec)

{

vec.at(0)=100;

}

int main() {

vector<int> vec{10,20,10};

for(auto i:vec)

{

cout<<i<<" -> ";

}

modify(vec);

cout<< " afetr"<<endl;

for(auto i:vec)

{

cout<<i<<" -> ";

}

}

**String in c++**

Two types :

C-styled string : char str[] <cstring> for strcmp(), strupr(),strlwr() ......

C++ string : string str ; // object <string> , .append(),.replace(),.at(),.find()....

//to get the string with spaces

getline(cin,str);

**C-styled string :**

#include <iostream>

#include <cstring> //similar to <string.h> in C

#include <cctype> // similar to <ctype.h> header file in C

// isdigit,ispunct,islower,isupper,toupper,tolower,isalpha,isalnum,ispace

using namespace std;

int main()

{

char str[] = "dineshKarthick"; // {'d','i','n','e','s','h','\0' };

char str1[] = {'d', 'i', 'n', 'e', 's', 'h'};

char str2[100] = "dineshraja"; //{d,i,n,e,s,h,\0,\0};

char str3[100];

strcpy(str3, str2);

cout << str3 << endl;

int result = strcmp(str3, str);

cout << result << endl;

int length = strlen(str);

// it gives -1 when , left ith character in string is less then

// right character in string, gives i for other case

// gives 0 in the case of equal string

strcat(str3, str);

cout << str3 << endl;

strupr(str3);

cout << str3 << endl;

}

**C++ string ( <string>):**

String in c++ are mutable , but in java ,strings are immutable

#include <iostream>

#include <string>

using namespace std;

// string is dynammic meanwhile c-styled string is static

// can be modified unlike string in java

/\* at(),length(),replace(),erase(),append(),insert(),compare(),substr(),find()

getline(cin,string),

\*/

int main()

{

string name = "dinesh";

cout << name << endl;

// concatenate using '+'

name = name + "raja";

cout << name << endl;

name = name + 'R';

name.append("Raja");

name.insert(10,"menaga");

cout << name << endl;

string lastname = "dineshrajaRraja";

//comparing with ==,....

// ==,<,>,!= can be used with strings unlike in java ,cannot use these

string result = (name > lastname) ? "true" : "false";

cout << result << endl;

//access using

cout<<name.at(2)<<endl;

cout<<name[2]<<endl;

//modify

name.at(2)='N';

name[3]='N';

// replace(startIndex,endIndex) similar to java but parameters will be indexs

lastname.replace(0,6,"DINESH");

cout<<"replace dinesh with DINESH : "<< lastname<<endl;

// substring

cout<<name.substr(0,2)<<endl;

// searching

// .find() is similar to indexOf() in java

cout<<lastname.find("raja")<<endl; // output index

cout<<lastname.find('R')<<endl;

cout<<lastname.find('raja',10)<<endl;// check after the index 10

//remove string

//similar to sb.delete(start,end)

lastname.erase(0,3);

cout<<lastname<<endl;

//length()

cout<<"length of lastname : "<<lastname.length()<<endl;

//get user input

string collegename;

cout<<"enter the college name without space"<<endl;

cin>>collegename; // similar to next() takes untill " "

cout<<"The college name is : "<<collegename<<endl;

string collegename1;

cout<<"enter the college name with space"<<endl;

getline(cin,collegename1);

cout<<collegename1<<endl;

return 0;

}

Dynamic allocation :

-> create use ='new' keyword

->we have to manually delete the dynamic memory using 'delete' keyword

**Pointers and References:**

int num=10

int &ref=num; //reference to num

int \*ptr=num //pointer

**Functions :**

1.return type

2. no return type

**NOTE : default argument in c++**

prototype of function(declaration)

pass by value

pass by referencepass by pointer

**this keyword:**

pointer to current object

Used to access the members of pointer object

**Static keyword**

static

-> static function

->static variable

->No static class in the direct way

**const**

-> with cannot change the value

-> can create const objects

const Myclass obj;

obj.setValue(3) // error due to const object

// to let the compiler to know that this setValue() will not modify the object members we do

void setValue() const

{

//implementation

}

**l-value and r-value:**

r-value in c++:

1. temporary object like obj=Myclass("hello");

2.value like int x=10 , here 10 is r-value

3. functions return none reference

string getName()

{

return "harry"; // r-value

}

**Access Modifier :**

NOTE : private by default , but in java , always public.

-> public =accessed outside the class

->private =no access outside the class

->protected = access outside the class by inside child class only

Constructor :

-> Default constructor (no arguments)

->parameterized constructor (with argument)

->copy constructor

invoked when :

Myclass obj1;

Myclass obj2=obj1;

Myclass obj2(obj1);

1.shallow copy for class with no raw pointers

2.deep copy for class with raw pointers

->Move constructor

to handle r-value as parameter

null out the source object and return this pointer

NOTE : Initialization list : this will be executed before the constructor body

Myclass(int data):value{data}

{

}

**Contructor calling :**

Mystring obj; // Default constructor

obj = Mystring("dinesh"); // Move assignment operator (if you explicitly create a Mystring object)

Mystring obj1 = obj; // Copy constructor

Mystring obj2 = "dinesh"; // Overloaded constructor

obj = obj1; // Copy assignment operator

MyClass obj1(10); // Normal constructor

MyClass obj2(std::move(obj1)); // Move constructor

- > child class dont inherit base class constructor ,assignment Overloaded

**Inheritence :**

**inheritance:**

Single Inheritance

Multiple Inheritance

Multilevel Inheritance

Hierarchical Inheritance

Hybrid Inheritance

class Derived :public Base

here Derived class is inherited as public

->inherited as public

public members of Base will be the public members of Derived class

protected memners of Base will be protected members of Derived class

private members of Base class has No accessed by Derived class

->inherited as protected

public and protected membesrs of Base class will be protected of derived class

private will be no accessable

->inherited as private

everything will be No accessable by Derived class

**Polymorphism:**

**compile time (static binding):**

Base obj=new Derived()//here it will be called based on the reference , not on object type

obj.display()// it calls the method of (base)

o overcome this ,we go for dynamic binding using virtual function

**runtime(dynamic binding):**

Base obj=new Derived()

obj.display() // it calls the method of object type (Derived)

**virtual function() override**

//above override is the specifier to make sure the compiler to overide that method

//it does not override , error will be thrown.

**Abstraction :**

done by using virtual function()=>0;

A class that has the virtual function with no implement will act as abstract class

A class that has the implementation of virtual methods in abstract class is Concrete class

where object can be created.

**Interface :**

done by pure virtual functions

A class that has only the virtual function act as interface

**Smart pointers :**

smart pointers

-> create dynamic object without using new

->no manual deletion using delete

types:

1.unique pointer

2.shared pointer

3.weak pointer

1.unique pointer:

->smart pointers cannot be copied and assigned

->can be moved

->dynamic binding

2.shared pointer

->multiple pointers can refer same memory

->smart pointers can copied and assigned

->can be moved

->dynamic binding

3.weak pointer

->to break the cycle scenerio in shared

ex: A refers B , B refers A

definition common for all:

for class:

unique\_ptr<Test> t2{new Test(2000,"test2")}

unique\_ptr<Test> t3=make\_unique<Test>(3000,"test3");

for data type:

shared\_ptr<int> p1=make\_shared<int>(1);

shared\_ptr<int> ptr{new int};

custom deletor:

to do extra work more that just delete the memory

**Streams:**

Base class: ios

child class: istream,ostream..

<iomanip>

-> to manipulating the cin,cout streams

functions :boolapha,setprecision(),fixed,setf(),hex..

filestream:

<fstream>

ifstream file.open("") // Input File Stream(ifstream)

ofstream file.open("")//output File Stream

flags like app

file.open("/path",iso::app)//app for open as appending

string line;

file>>line //read from file and store in line

file<<line //write from string line to file

/\*

open with mode :

ios::app //append

ios::in //read only

ios::out //write only

ios::trunc // clear content and open

\*/

**standard template library**

**Generic programming:**

**template:**

to make the function or class to handle any type of data

template<typename T> //declaration

T sum(T a,T b) {

return a+b;

}

**two templates:**

->class template

->function template

calling function:

sum<int>(1,3)

or sum(1,3) // this can handle any type

vector and all containers comes under template

**Generic using macros:**

#define // it is processed by preprocessor before compiler

**containers :**

it is collection of objects and can manipulate using using algorithm functions

like count(),copy(),find(),replace()

containers are

sequence container : array,vector,list,deque

Associative container : set,map ..

container adapters : stakc,queue,priority queue

to handle the list of user defined type (class ):

we should override the operators to use the sort(),find()..

example: we should override >,= < operators for sort() list<Person>

**lambda expression:**

it is function object.

ex: [](){cout<<"hello";}();

auto l=[](int a,int b){return a+b};

l(); //calling lambda ,here compiler create function object from this lambda

stateless lambda :

it does not access the members of the scope

ex:

int num=10;

[](){ /\*cannot access num \*/}

statefull:

int num=10;

[num]()mutable{num=num+10};//mutable allow the capture list variable to be modified

or

[=](){/\*can access all members\*/} //pass by value

[&](){}//pass by reference

[this](){}//pass by reference

can be used with algorithms functions:

example:

sort(vec.begin(),vec.end(),[](int a,int b){return a<b})

**Vector :**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

cout << "====================\n";

vector<int> list = {10, 11, 12, 13};

// modify the element using index and at()

list[0] = 34567;

list.at(2) = 34568;

// access using index

cout << list[0] << endl;

// can be accessed and modify using front() and back() methods

cout << list.front() << " " << list.back() << endl;

// add values using push\_back() method

list.push\_back(10000);

// insert element at particular position

int index = 4;

list.insert(list.begin() + index, 8786);

// pop the last element in vector

list.pop\_back();

cout << "last element removed" << endl;

// print size using size()

cout << "size of vector : " << list.size() << endl;

// remove element using index by erase()

list.erase(list.begin() + 2);

// iterations

cout << "iteration using for loop" << endl;

for (int i = 0; i < list.size(); i++)

{

cout << list.at(i) << " ";

}

cout << endl;

cout << "iteration using for each loop" << endl;

for (int i : list)

{

cout << i << " ";

}

// out of range exception will be thrown to the user

// cout<<list.at(100);

cout << endl;

vector<vector<int>> vector2d;

vector2d.push\_back((vector<int>){1, 2, 3, 4, 5});

vector2d.push\_back((vector<int>){2, 4, 6, 8, 10});

vector2d.push\_back((vector<int>){3, 6, 9, 12, 15});

cout << "2d vector elements" << endl;

for (vector<int> i : vector2d)

{

for (int j : i)

{

cout << j << " -> ";

}

cout << endl;

}

return 0;

}

**Access through :**

at() ,

vec[]

front()

back()

**Iterators :**

begin()

end()

rbegin()

rend()

**Modifers :**

Clear(),

Insert(it,value)

Insert\_range()

erase(it)

Push\_back() - append value

Pop\_back() - delete value

Swap()

vector<int> v1 = {1, 2, 3};

vector<int> v2 = {10, 20, 30};

// Swapping contents of vec1 and vec2

v1.swap(v2);

List : ( implemented as a doubly-linked list)

You can add and remove elements from both the beginning and at the end of a list, while vectors are generally optimized for adding and removing at the end.

Ex:

List .push\_back(),push\_front() ,pop\_back(),pop\_front()

**<algorithm>**

1. sort()   
 sort(vec.begin(),vec.end())

For reverse sort :

Sort(vec.rbegin(),vec.rend());

Also can, sort(vec.begin()+1,vec.end())

Since , begin() and end() returns iterators

2. find( list.begin(),list.end(),key)

3. max\_element(l.begin(),l.end()) , min\_element(l.begin(),l.end())

4. fill(l.begin(),l.end(),0)

Iterators : ( pointers )

// Create a vector called cars that will store strings  
vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};  
  
// Create a vector iterator called it  
vector<string>::iterator it;  
  
// Loop through the vector with the iterator  
for (it = cars.begin(); it != cars.end(); ++it) {  
 cout << \*it << "\n";  
}

**Set :**

A set stores unique elements where they:

Are sorted automatically in ascending order.

Are unique, meaning equal or duplicate values are ignored.

Can be added or removed, but the value of an existing element cannot be changed.

Cannot be accessed by index numbers, because the order is based on sorting and not indexing.

Add using : insert()

Set.insert(“dinesh”);

Delete element:

Set.erase(value)

Set.emty()

Set.find()

Size()

Count()

**Map**

A map stores elements in "key/value" pairs.

Elements in a map are:

Accessible by keys (not index), and each key is unique.

Automatically sorted in ascending order by their keys.

|  |  |
| --- | --- |
| Vector | List |
| Vec.size() | List.size() |
| Vec.push\_back() | List.add() |
| Vec.pop\_back() | List.remove(list.size()-1) |
| Vec.erase(vec.begin()+index) | List.remove(index) - int index  List.remove(value) - Integer value |
| Vec.insert(vec.begin()+index, value) | List.add(value,index) |
| #include<algorithm>  Auto it=find(vec.begin(),vec.end());  If( it!=vec.end()) | List. contains(value ) = return true/false  List.indexOf()  List.lastIndexOf()  Collections.binarySearch(list,element) |
| Vec.insert(vec.end(),vec2.begin(),vec2.end()) | List.addAll(list2)  Collections.addAll() |
| vec.erase(vec.begin(),vec.begin()+2); | List.removeAll(list2)  Collections.removeAll() |
| replace(vec.begin(),vec.end(),key,value) | Collections.replaceAll(list, "apple", "orange"); |
| Vec[index]  Vec.at(index) | List.get(index) |
| Vec[index]=value  Vec.at(index)=value | List.set(index,value) |
| Vec.clear() | List.clear() |
| Vector<int> vec(vec1.begin(),vec2.end())  Vec2.assign(v1.begin(),v.end()) | List2=list.subList(index,index) |
| Sort(vec.begin(),vec.end() comp) bool com(int a,int b)  {  Return a<b;  }  sort(vec.begin(),vec.end())  For reverse sort :  Sort(vec.rbegin(),vec.rend()); | Sorting:  List.sort()  Collections.sort(list)  Collections.sort(list,Collections.reverseOrder())  Collections.reverse(fruits); |
| equal(v1.begin(), v1.end(), v2.begin()) return true / false | List1.equals(list2) |
| Include<algorithm>  max\_element(vec.begin(),vec.end()) | Collections.max(list) |
| Include<algorithm>  min\_element(vec.begin(),vec.end()) | Collections.min(list) |
| Include<algorithm>  fill(l.begin(),l.end(),0) | Collections.fill(list,0) |
| Include<algorithm>  count(vec.begin(), vec.end(), value); | Collections.frequency(list,value) |
| copy(source.begin(), source.end(), destination.begin()); | Collections.copy(destination, source); |

Strings:

|  |  |
| --- | --- |
| String is object in C++  <string> Mutable | String is an object in java  immutable |
| string str=”dinesh”; | String str=”dinesh”; |
| string str=new string(“Dinesh”) Dynamic creation | String str=new String(“dinehs”) |
| Str.at(index)  Str[index]; | Str.charAt(index) |
| Str.at(index)=value  Str[index]=value | **Immutable** |
| Str.find(str2)  If(str.find(str2)==-1) then , no match  s.find(sub, 0, 4); with range | String.indexOf()  String.contains();  String.lastIndexOf() |
| stringstream | Str.split() |
| String stream | String.join(array, “,”) |
| transform(s.begin(), s.end(), s.begin(),  ::toupper);  for (int i = 0; i < s.length(); i++)  s[i] = toupper(s[i]); | Str.toupperCase()  Str.tolowerCase() |
| string r = s1.substr(3, 2); | Str.subtring(startIndex,endIndex)  Note : EndIndex-1 |
| Str1+str2  Str1.append(str2) | Str1.concat(str2)  Str1+str2 |
| Str1 = = str2 | Str1 == str2 ; note: only for comparing literals  Str.equals(str2)  Str.equalsIgnoreCase(str2) |
|  | Str.startswith()  Str.endsWith() |
|  | Str.isEmpty() |
| str1.replace(pos, n, m, c) // Replace with character str1.replace(pos, n, str2) // Replace with string str1.replace(pos1, n, str2, pos2,m) // Replace with substring str1.replace(first, last, n, c); // Replace Character str1.replace(first, last, str2) // Replace String str1.replace (first, last, str2\_first, str2\_last); // Replace Substring  str1: String in which we have to replace the multiple characters.  pos: Index to the position in str1 where we have to start replacing the characters.  n: Number of characters which we have to replace.  m: Number of times we have to repeat the single character.  c: Character by which we have to replace. | Str.replace(target,replaceValue)  Str.replaceAll(regex) |
| strcpy(arr, str.c\_str()); | Char arr[]=Str.toCharArray() |

String buufer (thread safe )or String builder(no thread safe) – mutable strings:

StringBuffer sb=new StringBuffer(str);

Sb.append(str);

Sb.insert(index,str)

Sb.delete(index,index)

Sb.replace(index,index,str)

Sb.reverse()

Include<cctype> vs Character Class in java

|  |  |
| --- | --- |
| toupper(ch) ,tolower(ch), isupper(),islower() | IsUpperCase(ch),isLowerCase(ch),toUpperCase(ch) |
| isalnum(), isalpha() | IsLetter(ch),isLetterOrDigit(ch) |
| ispunct() | - |
| isspace() | IsWhitespace(ch) |
| isdigit() | isDigit() |

C++ static library creation and use in Visual studio :

[Walkthrough: Create and use a static library (C++) | Microsoft Learn](https://learn.microsoft.com/en-us/cpp/build/walkthrough-creating-and-using-a-static-library-cpp?view=msvc-170)

Static file include:

1. Properties

C/C++ -> Additional Include Directories : add path where the .h files are placed

Linker -> General -> Additional Library Directories : add path where the .lib file present

Input - > Additional Dependencies : Add the library name with .lib extension after we add the .lib file in the sln working directory in VS