

- * Competitive Coding
- * Stack Overflow
- * Git & GitHub
- * More Practice.

Machining Solutions

- * Hack-a-ton Requirements.
- * Competitive Coding.
- skills Req.
- * Projects platform,
This to cover till
the end of month

Machining Solutions

C++
STL
DSA

Machining Solutions

char : 1 byte

char16_t : 2 byte (16 bits)

char32_t : (32 bits)

wchar_t : largest avail. set.

sizeof: determines the size in byte of a type or var.

<limits.h>

<cfloat> include file contain size & precision information

* Vectors: Dynamic Array:

include <vector>

vector <char> vowels;

vector <int> test;

→ Other way to initialize.

vector <char> vowels(5);

vowels { 'a', 'e', 'i', 'o', 'u' };

→ Same as array

cout << test->vowels[0];

vowels[0] = 'a';

cout << test->vowels.at(0);

test->vowels.at(0) = 'o';

int main() {
 print-vector(name);
 // + int values.
 // displays all element
 <cout> <endl>
 // the vector.
 // the component given
 // declaration
 // instant. this
}

// Constructor initial...

name { 'a', 'e', 'i', 'o', 'u' };

cout << name[0];

name[0] = 'a';

cout << name.at(0);

name.at(0) = 'o';

cout << name[0];

name[0] = 'a';

cout << name.at(0);

name.at(0) = 'o';

cout << name[0];

name[0] = 'a';

→ * push-back () : used to add element in {}
pop-back () : deletes last element in vector
Ex: test-scores.push-back(90); vector been {100, 95, 90} added
→ # That's the advantage.
Sort:
sort (vector-name.begin(), vector-name.end());

Size : test_scores.size() // shows no. of elements.
array [] {1, 2, 3, 4};

* 2-D vectors: (Ex. 59) (using function)

```
vector<vector<int>> movie_ratings  
{  
    {1, 2, 3, 4}, // 2 total  
    {1, 2, 3, 4}, // 2 total  
    {1, 2, 3, 4}  
};
```

int score [] {100, 10}

int score [2] { }
→ 2 elements
are initialized
to zero.

```
cout << movie_ratings[0][0]; instead  
or  
cout << movie_ratings.at(0).at(0);
```

```
cout << (num1 == num2) // 0 or 1  
cout << std::boolalpha; // True or False  
#include  
#include  
bool result {false};  
cout << boolalpha // will display T/F instead  
of 0 & 1
```

Section 8

Ex:

101

a % 100

1

$\frac{101}{100} = 1$

Vector:

v.erase(v.begin() + 4) (removes 5th element) b = 101 - 1 x 100

v.erase(v.begin() + 2, v.begin() + 5) (Removes elements b/w
3 & 6.)

Controlling Program Flow

Conditional Operator : (?:)

switch (ch) {

 case exp1 : state ;
 break ;

 case exp2 : state ;
 break ;

 ...

 default :

 }

Ex:

 case 'a' :
 case 'A' : cout << "90" ;

 cout << "90" ;

 case 'b' :

 case 'B' : cout << "80" ;

 cout << "80" ;

* (cond-exp) e.g. exp1 : exp2

Ex: 101 result = (a>b) ? a : b

Output : cout << ((num1 > num2) ? num1 : num2);

Additional operators - (& (address-of), &* (derefer.)) are used.

Ex: & (address-of operator) begin for (i = 1; i < 10; i++)

vector loop: maintains ref. how it is done

```
vector <int> nums {10, 20, 30, 40, 50};
```

```
for (int i{}; i<nums.size(); ++i)
```

```
    cout << nums.at(i) << endl;
```

```
; ( ) do
```

Range-based for loop:

```
for (var-type var-name : sequence) (i: i is)
```

```
statement;
```

```
(i) maps onto
```

auto keyword: by auto checks the data type.

```
int scores[] {100, 99, 98};
```

```
for (auto score : scores)
```

```
    cout << score;
```

```
{
```

Eg: for (auto temp : {60.2, 70.1, 80.3}) iteration

```
sum += temp;
```

```
++size;
```

```
[ ]
```

or last = T

last = A

new value = H

#include <iomanip>

or billions = 2

```
cout << fixed << setprecision(number);
```

Ex: cout << fixed << setprecision(1) << 4;

or: cout << temp;

↙ output: 4.0

after decimal

while: { * used to check the wrong entry }

char ch = 'A';
while (ch != 'Q') {
 cout << "Enter a character: ";
 cin >> ch;
 if (ch == 'A')
 cout << "Hello World!" << endl;
 else if (ch == 'B')
 cout << "Good Day!" << endl;
 else if (ch == 'C')
 cout << "Good Evening!" << endl;
 else if (ch == 'D')
 cout << "Good Night!" << endl;
}

{ return 0; }

* do while : // used for continuation

```

do {
    // do something
} while (condition);
while ( );

```

Infinite Loop:

```

for(;;) {
    while (true) {
        char again {3};
        cout << "DO you want to cont. Y/N";
        cin >> again;
        if (again == 'N' || again == 'n')
            break;
    }
}

```

Challenge (fizz, buzz, cond) obj. E

P - Print No.

(print = + endl)

A - Add "5 added"

[1, 2, 3,]

M - Display mean

"[] - the list is empty

S - Smallest no.

<min() > default //

L - Largest No. (max() > default > true)

(max() > true > false)

or <max() >

(max() > true > false)

include <bits/stdc++.h> others *

for Min. Element :

cout << *min_element(a.begin(), a.end());

{a is vector.}

String : #include < cstring.h> for string function.

#include <cctype> - C type string
#include <string>
isalpha(c), isalnum(), isdigit(), islower(), isupper(),
isprint(c) - True if c is printable characters.
ispunct(c) - punctuation
isspace(c) - whitespace.
cin >> name;
cout << name; // Hitesh Gorantla

cin.getline(name, 50); // stops when it detects whitespace
cout << name; // Here it displays Hitesh Gorantla

size_t if{} : unsigned int. { only +ve values }
C++ type String
string s1; // Empty
" " s2 {"Hitesh"}; // Hitesh
" " s3 {"Hitesh"}; // Hitesh
" " s4 {"Hitesh", 3}; // Hit
" " s5 {s3[0, 2]}; // Hitesh
" " s6 {3, 'x'}; // XXX

string part1 {"C++"} \Rightarrow stoi() : to check entered string is an int.

string part2 {" is p."}

s = part1 + " " + part2 + " language"
// C++ is p. lang.

s = "C++" + "p." // illegal.

$s2[0] = 'H'$ Hitesh

$s2.at(0) = 'P'$: pitch

`substr()` - // Extracts the sub-string from the string $s1$ { "This is a test" } - (actual string)
 cout << s1.substr(0, 4); // This
 cout << s1.substr(5, 2); // is

`find()` - to find in the string. } // return > index

cout << s1.find("This"); // 0 starting location
 cout << s1.find("is"); // 2 value.

{ new string + strings } this becomes : (8) is 2 is

{ cin >> s1; cout << s1; // Hello still get " space. }

getline (cin, s1); // Read complete line till "\n"

cout << s1; // Hello Hitesh

Remove ch: - `erase()` `clear()` to empty the string

cout << s1.erase(0, 5) // Hitesh

index to where } drop private

cout << s1.erase(7, 2) // Hello Hitesh Hello Hitesh

specify + string + = + stop & &

drop off is ++ }

Hello + stop + ++ = 8

```

#include <cmath>           // math operations.
#include <cstdlib>
#include <cstdlib>          // srand & rand
                           // rand() generates random no.
                           // srand() seeds pseudo rand.
                           // rand() generates random no.

#include <ctime>           // required for time()
                           // srand(time(nullptr));
                           // sqrt() // square root
                           // cbrt() // cube root
                           // ceil() // bin // nearest int. value (next value)
                           // floor() // returns largest int value
                           // Eg: 74.12 gives 74
                           // round() // rounds off
                           // pow(2,3) //  $2^3$ .
                           // cout << static_cast<double>(total) / number.size();
                           // to change the data type.
                           // int
                           // vector

```

Function Prototypes

Eg 1:

```
#include <math.h>
int area() {
    // calculate area
    // return ans
}
```

```
int vol() {
    // calculate volume
    // return ans
}
```

```
int main()
```

Eg 2:

```
#include <math.h>
void area();
void vol();
int main()
```

void area();

void vol();

int main();

(return type) \rightarrow void \rightarrow no return value.

values too logical variables \rightarrow (variables)

// default values should be declared in function
 \rightarrow prototype.

Eg: // double calc_cost (double base = 100);

int function() // have return value

void function() // no return value.

return

return value

no return value.

Overloading Function

int add (int, int) *// for integer*
 double add (double, double) *// for float*

{ option } step. tri + tri *// stepwise tri*
 to call the function *(my code + add)*

// static addition (float) *// 1*

// add (2, 2); *// correct*

// add (2.2, 2.1); *// correct > robbery*

Scope Rules:

void sum () { *{ for more tri }*

static int num {5}; *{ for more tri }*

num = 10; *num* retains the value obtained
 from the assignment after the operation. *{ for more tri }*

Inline function:

// inline int add_no (int a, int b) *{ for more tri }*
 void main () { *{ for more tri }*

+ faster

- could cause

code bloat

why? *generates*

inline assembly

Recursive function:

// shallow step tri + tri

// repetitive *? /*

definition of multiplication

1000 = step tri + tri

Pointers: It is a variable ~~without pointer~~

↓
It holds the address of the data types.
variable-type * pointer-name;

int *intptr; int *intptr { nullptr } ;

char * char_ptr; ~~initially null~~ // initialized 0,

// should initialize pointer //

otherwise it will have some garbage value

vector <string> *p4 { nullptr }; ~~initially empty~~

int num { 10 };

{

cout << num; ~~initially 10 then jidatka~~

Pointers can be interchangeable used as array.

int score { 10 };

int *score_ptr { nullptr }; ~~initially 0~~

score_ptr = & score;

cout << score; ~~we take the value~~ // value stored

cout << & score; }

cout << score_ptr; ~~initially 0~~ // location of score

Dereferencing a pointer:

Advanced notes

int *intptr { nullptr };

& : used to get

* intptr = 200;

location of variable.

Dynamic Memory Allocation:

```

int size;    double * temp_ptr {nullptr}; // points to heap
temp_ptr = new double [size]; // stored in heap.
cout << temp_ptr; // prints address
delete [] temp_ptr; // deletes the element

```

Subscript Notation

array-name [index]
pointer-name [index]

Offset notation:
 $\Rightarrow \text{*(array name + index)}$
 $\Rightarrow \text{*(pointer name + index)}$

$\text{* score_ptr} \neq (\& \text{score}) \& \&$

int * score_ptr {& score};

// then $\text{* charptr} \{ \& \text{name}[0] \}$;

const int * score_ptr {& score} // add. PA // score = 100; can be changed but not value

int * const score_ptr {& score} // add. same, value

const int * const score_ptr {& score} // add. both can't be changed

Reference:

- An alias for a variable
- Cannot be nullified (cannot be initialized to nullptr)
- Once initialized can't be refer to a different variable.

int & ref {num}; // num = 100;

OOPs:

class name
 {
 // attributes

and methods } [+] attributes are static methods

// methods [+] methods are functions

+ methods with return type } [+] return type [+] methods

 + methods with parameters

* (* frank account).balance ;

(* & * & *)

(return type) frank-account → balance ;

initialise type [+]

[+] return type [+]

[+] return type [+]

* To create object:
 + (classname *) = straightforward

name Hitachi;
 ↓
 classname → object.

Implementing Outside Class:

class Account {

 { frank } step-weise of the lines

private: { frank } step-weise lines to the

 accessories

protected and public: { some } step-weise lines of the lines

 void set (double bal);

};

: some { }

 functions with it

void Account::set_balance (double bal){}

 { some } step-weise lines of the lines

 { some }

 { some }

Include Guards

```
#ifndef _ACCOUNT_H_
#define _ACCOUNT_H_
```

}
#endif

OR

pragma once

Account.cpp

Account.h

main.cpp

include class header file.

// Some compilers supports

// function prototype declaration
// with #include "Account.h"

// includes the class & function prototype.

Constructor & Destructor:

- Can be overloaded

- Can't be overloaded

- Many

- only one in class

- Invoked auto.

- Invoked auto.

- when objects are created.

- when objects are destroyed

Player Player + enemy = new Player;
enemy → set-name ("Enemy"); // values need to
be provided.

With constructor Name

Overloaded Constructor:

Players();

Players (string name, int n);

strapping constructor //

Creating Objects:

Players empty;

// Name, 0, 0

Players hero ("Hero")

// Hero, 0, 0

* Players + enemy = new Player ("Enemy", 1000, 0);
negating constructor

// Enemy, 1000, 0

Better way:

Players::Players() // constructor is robust

: name {"Hitesh"}, health{0}, xp{0}

destructor is robust

destructor is robust

sets of 200 lines

best

Delegation Constructor:

Player::Player(): // code for one constructor can
call another in the
initialization list.

}

Copy Constructor:

Type:: Type (const Type & source);

Base b{100};

Base b1{200};

Shallow Copy

Pass object by value

Return object by value

One object based on another (copy)

Deep Copy

- Each copy points to

same storage in heap

"d1obj1" = address of object

in different

class date member as pointer.

Deep :: ~Deep()

Therefore if we change data in base class it will affect all objects created from shallow copy.

Move constructor:

```
int x{100}; // Ownership given to l
int & l-ref = x; // l reference
l-ref = 10; // move if x to 10 inside l
```

```
int & r-ref = 200; // R reference
r-ref = 300; // change to 300;
```

Syntax:

Type:: Type (Type & source); // Prototype

(~~copy~~ keyword) Move (Move & & source) body; }

int & data { source.data } { return & data; }

source.data = nullptr;

cout << "Move constructor-moving resource:" << data;

}

this pointer:

introduction pg 5

It is used to access value of members of another class from one class to other.

```

→ Class Student {
    string Name;
}
Struct :
```

class student {
 ~~public~~
 this->Name = "Hitesh";
};

- Variables declared inside struct are public by default.
- Don't declare methods in struct.

```

class Account {
    std::string get_name();
}
Account::get_name() { // prototype or tri
    std::string Account::get_name(); // or = func def tri
}
Account::get_name() { // or = func def
    std::string Account::get_name(); // 1) definition outside
}
Account::get_name() { // 2) definition inside class.
}
```

Friend of a class:

```

class Player {
    friend void other_class::display(Player &p);
    friend class other_class; // Player class has
    public : // other-class has access to Player class.
}
```

! Static: class members:

Static function have only access to static data members.

Operator Overloading

Overloading the copy assignment operator (Deep copy)

Contract

Type & Type	operator = (const Type &rhs);	// copy
Type & Type	operator = (Type &&rhs);	// move

// bad operator = = (const MyString &rhs) const;

Operator Overloading Global:

Number operator + (const Number &lhs, const Number &rhs);

Stream insertion operator (<<):
 std:: ostream & operator << (std:: ostream &os, const MyString &obj) {
 os << obj; }
 always link to iostream library
 without brief code

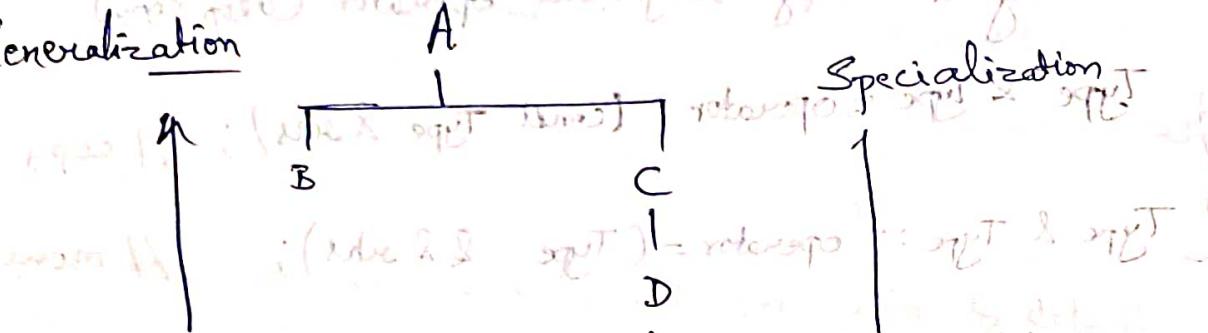
* Inheritance: ("is-a") ~~concept~~ Composition "has-a"

Base class (Super class) and derived class

Derived class (child class)

(most) inheritance goes with generalizations

Generalization



Combining similar (child & parent) classes = inheritance

classes

Specialization

(Creating new classes)
from existing classes

members from child members + members from parent
Protected members:

- Not accessible by objects of Base or Derived

types, as & new to bts => protected members etc.

Derived class does not inherit:

- Base class constructor & destructor
- " " overloaded assignment operators
- Base class friend function.

class Base {

}

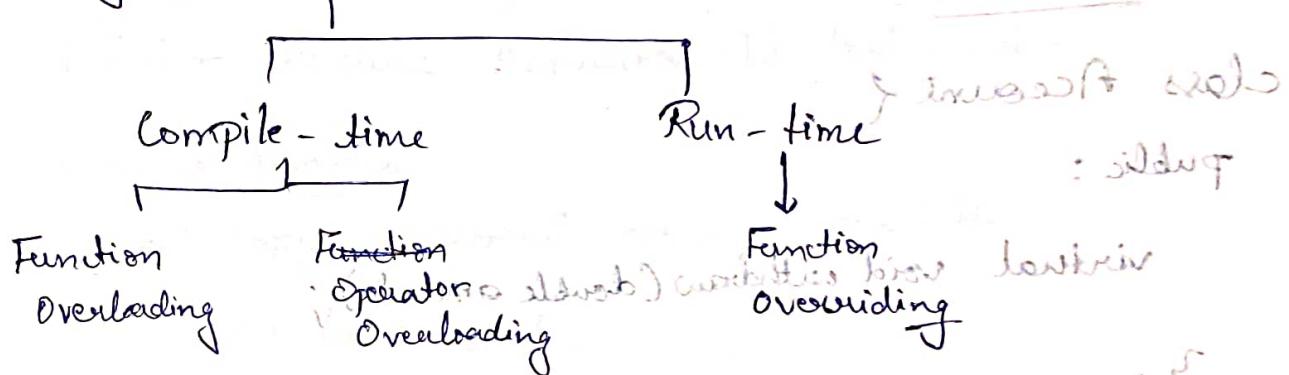
Class Derived : public base {

using Base::Base(); // Base class constructor
accessible in derived
(because it's a friend of class).
(over)loading → no

Derived () : Base {}, double_value{0}

{}
{}
(it's possible to call Base class
constructor called
(over)loading ← (it's good) in derived

* Polymorphism:



Static Polymorphism:

void greeting (const Base & obj) {

cout << "Greetings";
obj.say-hello();

}

int main() {
base b;
greeting(b);
}

// Static Binding

Dynamic poly.

- Inheritance
- Base class pointer
- virtual functions

```
vector<Account*> accounts { p1, p2, p3, p4 };
```

for (auto acc : accounts)
 acc->withdraw(1000);

or

```
Account *array[] = { p1, p2, p3, p4 },
```

for (auto i = 0; i < 4; ++i)

array[i]->withdraw(1000);

Virtual Function :

```
class Account {
```

public:

virtual void withdraw(double amount);

}

Virtual Destructor:

```
virtual ~Account();
```

→ No virtual constructor are there.

final class: Prevents further overriding.

class My final {

y;

class der: public my { // Error compiler
};

Abstract Class: // cannot have objects.

Pure virtual function:

virtual void function () = 0;

Smart Pointers: C++11.

#include <memory>

RAII - Resource Acquisition Is Initialization.

Unique Pointers:

Not Copy assignment operator available.

Can't be moved.

unique_ptr - creating, initializing & using

unique_ptr<int> p1{new int{0}};

p1.get() // address

p1.reset() // nullptr now

if (p1)

cout << *p1; // if p1 is initialized
then it will execute.

C++

* SMART Pointers

- Unique ptr
 - Shared ptr
 - Weak ptr
 - Custom ptr.

```

vector<unique_ptr<int>> vec;
    " " <int> ptr {new int {100}};
vec.push-back (ptr); // Error - copy
vec.push-back (move(ptr)); // Use this.

```

```
// std::unique_ptr<int> p1 = make_unique<int>(100);
```

Shared pointers:

~~std::shared_ptr<Test> p;~~

new integer makes here <int> pi { new int {100} }.

```
// shared_ptr<int> p1 = p1.make_shared<int>(100);
```

Weak Pointers:

```
std::weak_ptr<int> a_ptr;
```

II declaration

11 to call destructor we use

weak ptr with shared ptr.



Custom deleters

Functions

Lambdas

```

shared_ptr<Test> ptr(new Test {100}, [] (Test *ptr)
{ cout << "It Using Custom deleter" << endl;
    delete ptr;
});
```

Function:

```

int main()
{
    shared_ptr<Test> ptr1( new Test{100}, my_deleter );
    //> this is the copy constructor
    //> when we do this, it's like we're
    //> creating a new object & then
    //> giving it to another object to use
    void my_deleter(Test *ptr)
    {
        //> when this goes out of scope, it's
        //> deleted
        delete ptr;
        //> this is the destructor
    }
}

```

(if this is copied, then = 19 < this > dtg-upin n: b72)

Exceptional Handling:

C++ Syntax:

- throw if cerr is same as cout
if (foo) throws; it's not but used when program
crashes to print statement.
 - try (os) < try > blocks when try = 19 < try > dtg-blocks
 - catch
- without break

not break

if (p) < try > dtg-blocks :: dtg

also see robustness (no) ok

dtg-blocks allow try down

multiple methods

subsequent assignment

(left first) [], (last first) try < try > dtg-block

{} blocks > "robust" method give it "is true" if

dtg-block

{}

I/O AND Streams

ios, ifstream, ofstream, fstream, string stream.

Boolean Type:

cout << boolalpha; // T or F

cout << noboolalpha; // 0 or 1

std::cout << std::setf(std::ios::boolalpha); // T or F

std::cout << std::resetiosflags(std::ios::boolalpha); // 0 or 1

Integers:

- dec // no shift
- noshowbase // no shift
- no uppercase // no shift
- noshowpos // + on the intermediate part

Floating Point:

- setprecision // precision
- fixed // default precision is '6'
- noshowpoint // to display zeros.
- no uppercase
- noshowpos

std::cout << std::resetiosflags(std::floatfield);

Field width, align and fill:

- setw
 - left-justified by default; justification is right
 - fill // to fill empty space.

Ex: std::setfill ('-'); cout << two;

`for (int i = 0; i < 100000000; i++)` ; style based on >> func

fstream & ifstream class // without opening a file

- File opening a file
• File opening a file

Output file :

(only good with lots) reflects two sides

fitream & ofstream

```
out-file.open (filename) ;    // user is putting the  
out-file.is-open ()      // file open file name. successfully.
```

String Streams: strm no + strm : $\text{cout} \ll \text{strm}$

#include <iostream>

'S' ai nezzigreq illeqek.

water depth of 11 trophic strata.

153

۱۰۷

she was .

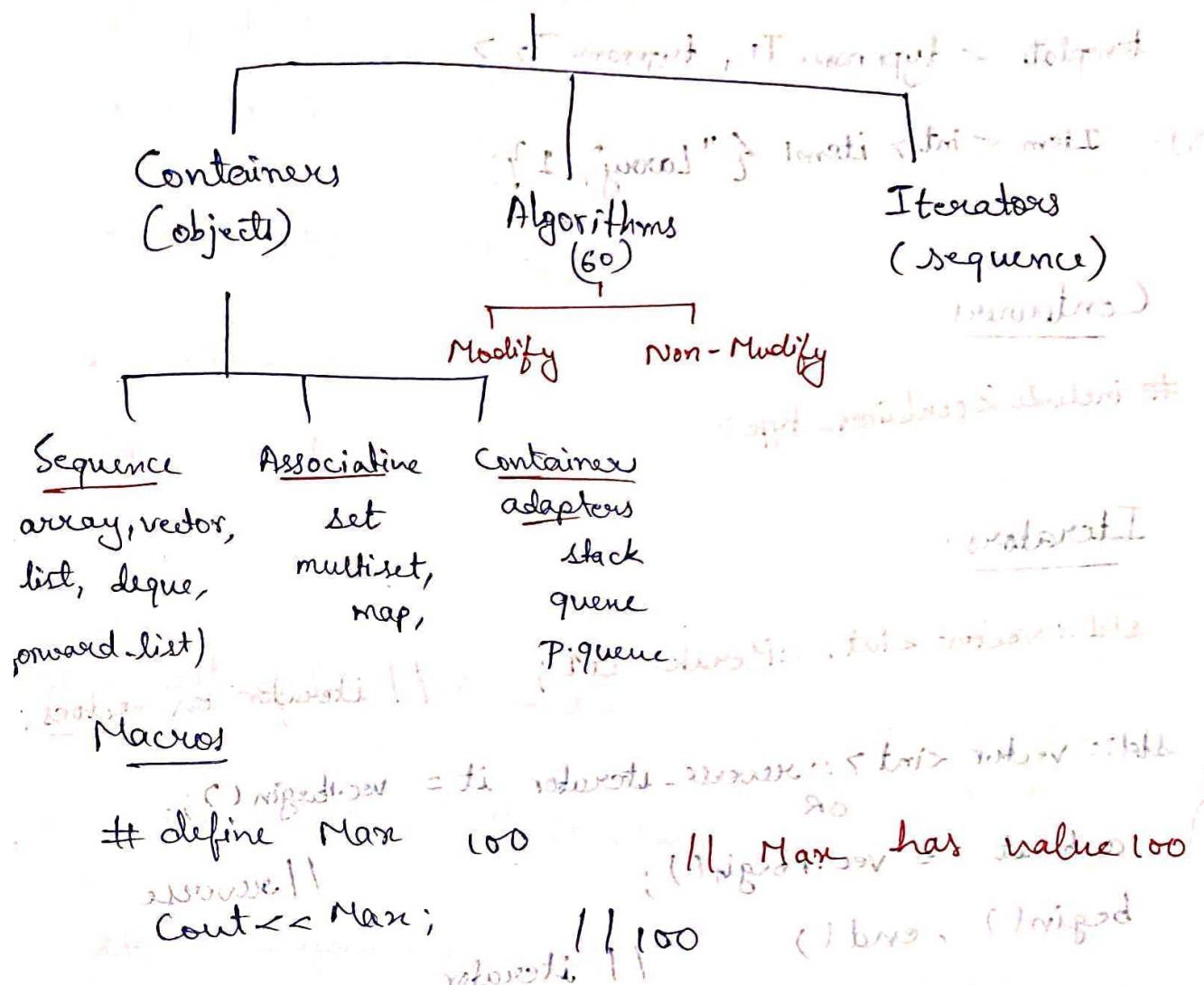
卷之三

186

(b) gift to all children before the testator dies

STL

std::list (class)



Macros

```

#define Max 100
cout << Max;
  
```

(Max = 100) has value 100

(Max, 100) (Max, 100)

Template: template<typename T>

```

class <=> typename T {
  
```

Eg: template<typename T>

```

T min (Ta, Tb) {
  
```

return (a < b) ? a : b;

}

(100, 200) (100, 200)

(100) (100)

(100) (100)

Class Template:

template < typename T₁, typename T₂ >

Eg: Item < int > item1 { "Larry", 1 };

int name

(name)

Containers :

include < container-type >

Iterators:

std::vector < int > ::iterator it2;

std::vector < int > ::reverse_iterator it = vec.begin();

OR

auto it = vec.rbegin();

begin(), end() // iterator

<begin(), cend() // const_iterator

rbegin(), rend() // reverse_iterator

crbegin(), crend() // const_reverse_iterator.

d: d > 0 (d > 0) number

Algorithms

#include <algorithm>

- `find()` // find the position of element
- `count()`

~~algorithm::find(,)~~ { } ; ~~int l, b; }~~ } count(~~the~~ no. of times present.

- `count_if()` // Conditional count - X expression

- `replace()`

Eg: `replace(vec.begin(), vec.end(), 1, 100);`

o of elements No 1 will be replaced by 100

- `all_of()` // condition same for all element in vector.

- `transform()` // to change the case (Upper/Lower)

`std::transform(str.begin(), str.end(), str.begin(), ::toupper)`

add scope
start end (str.begin(), str.end())
(oi, di) true -> (oi, di) false ->
Printing Global scope

- `adjacent_difference()` // if -at -divides

~~if (true) {
auto arr1 = {1, 2, 3, 5};
auto arr2 = {1, 2, 1, 5};
auto adjacent = std::adjacent_difference(arr1.begin(), arr1.end());
for (int i = 0; i < arr2.size(); i++)
if (arr2[i] == adjacent[i])
cout << "found" << adjacent[i];
}~~

~~if (adjacent[i] == arr2[i])
cout << "found" << adjacent[i];~~

#include <numeric>

- `accumulate(b, e, 0);` // sum to find if 10 it will add 10 to total.

4. Sequence Container - Array

#include <array>
double arr<C++11>
double braces in var <C++11>

std::array<int, 5> arr1 {{1, 2, 3, 4, 5}}; // initialization

cout << arr.front(); // 1
cout << arr.back(); // 5

- empty() // 0 or 1 if empty
- max_size() // (size of array)

arr.fill(0); // fill all element to 0.

arr.swap(arr1); // arr1 arr \leftrightarrow arr1 = 0

Vector: set elements ok // most work

max_size(); // (size of array) size of array vec

* auto it = std::find(vec.begin(), vec.end(), 3);
possible

std::vector.insert(it, 10); // inserts element before 3rd.

• shrink-to-fit(); // reduces the capacity

→ std::copy(vec1.begin(), vec1.end(), std::back_inserter(vec2));

v.1
* std::vector t1{1, 2, 3, 4, 5};
{ 1, 2, 3, 4, 5 }
{ 1, 2, 3, 4, 5 }

vec2 { 10, 20 }; → inserts 10 at back
copy of element

std::copy(vec1.begin(), vec1.end(), std::back_inserter(vec2));
start to end
{ 1, 2, 3, 4, 5 } { 10, 20 } { 1, 2, 3, 4, 5, 10 }

→ std::vector // output
{ 10, 20 } { 1, 2, 3, 4, 5 } ...

last of take New in v1

3. STL: deque

#include <deque>

```
std::deque<int> d {1,2,3,4,5}           // initialization
d.push_back (6)    // {1 2 3 4 5 6}
d.push_front (6)  // {6 1 2 3 4 5}
```

deque is a container which stores elements in a contiguous block of memory.

`push-back`: // {1 2 3 4 5 6} // back_inserter

`push-front`: // {6 1 2 3 4 5} // front_inserter

4 list & forward-list (Seq. containers)

#include <list> // bi-directional

- In list subscript `4 - at (0)` is not available.

Person p1 {"Hitesh", 18};

l.push_front (p1);

l.pop_front(); // deletes first element.

std::list<int> l {1,2,3,4,5};

auto it = std::find (l.begin(), l.end(), 3);

l.insert (it, 10); // 1 2 10 3 4 5

l.erase (it); // erased 3, 1 2 10 4 5

l.resize (2); // 1 2

l.resize (5); // 1 2 00 0

#include <forward_list> // one direction.

// single linked list

• only front();

• push_front();

f-list<int> l {1, 2, 3, 4, 5} {2) push-front.b
{3) push-back.b
{4) insert-after(it, 10);

l. insert-after(it, 10); // 1 2 3 10 4 5

l. emplace_after(it, 100); // 1 2 3 100 4 5

l. erase_after(it);

l. resize(2); // 1 2

std::advance(it, -4); // point to position -4.

list<Person> s; if (s.empty()) {Person}

std::s.emplace_back("Hitesh", 50); // no need to
create object.

{front(), back(), size(), tail::data}

(8, 1) front(), (Hitesh, 50) back(), size(), tail::data

STL Set Containers: (Binary tree or hashsets)

#include <set>

{ No, duplicate element allowed }

* no front & back, \Rightarrow first value.

* .at() & [] \Rightarrow A.

* s.insert(7); adds 7 in set in order.

s = {1, 2, 3, 4, 5}

s.erase(3); // erase 3

auto it = s.find(5); // location of 5.

num = s.count(1); // 0 or 1.

s.clear(); // removes all

s.empty(); // True or False

Sets: set \rightarrow mainly used -

unordered_set — allow dep.

multiset — allow duplicates [elements]

unordered_multiset — no \Rightarrow / unordered set

values: value1, ..., "value" = [no reverse iteration].

• s.emplace(,);

set_name

value

STL - Map (Associative containers)

#include <map>

- Key , value pair (No. duplicates)

- { } [] & { } to *

std:: map < std:: string, int > m; { } to *
value at key & key

{"Hitesh", 18}, { } to *
pair, set

{"GS", 18}, { } to *
pair, set

};

{ } to * to * () to * to *

- No. front & back

• m.insert (p1); // p1 {"Hitesh", 18};

• m.insert (std:: make_pair ("Hitesh", 18));

or

m["Hitesh"] = Student();

update ["Hitesh"] = "None"; // update value

- Rest same as set

Stack :

#include <stack>

int > stack<int> s1;

- push
- pop
- top
- empty
- size

Last-in-first-out - ~~first-in~~

last in last out

stack <int> s2; // deque

stack <int, vector<int>> s3; // vec.

(")



Queue :

First-In-First-Out

No iteration support.

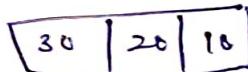
#include <queue>

- | | |
|---------|-------------------------|
| • push | • front - front element |
| • pop | • back - back " |
| • empty | |
| • size | |

queue <int> q; // deque

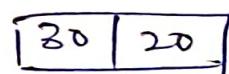
queue <int, list<int>> q2; // list

cout << q.front(); // 10



cout << q.back(); // 30

q.pop(); // remove 10



Priority Queue

* heap is used behind

* Same as queue with

well (POP is \leftarrow (remove) \rightarrow (decrease))

• top (n)



stack

stacks & stacks II

stl:: priority_queue <int> p;

get

push

size

empty

two - hand - wI - hand

programme without error

source > student #

two - hand - hand - hand

w hand - hand - hand

if (p < t) give

else

swap II if p > t swap

tail II if p < t swap, tail > swap



01 II if (tow).p > tail

02 II if (head).p <= tail



or worse II

(()q.pop()

if (swap == 1)