C++ [redacted] NOTES

* Two ways to initalize Constructors

i) Using Member initialization list ← only method for initializing refrence members

```
Public:
  Point(int i=0, int j=0): x(i), y(j) {}
```

ii) Using assignment

```
Public:
  Point(int i=0, int j=0) {
    x=i;
    y=j;
  }
```

* Inheritance constructors

```
Class area{
protected:
  int length;
  int breadth;
public:
  area( int l=0, int b=0) : length(l), breadth(b) {}
};
Class volume {
protected: private:
  int height;
public:
  volume (int h=0, int l=0, int b=0): [redacted] area(l,b), height(h){
  void getVdume(){
      (out << length * breadth * height <<endl;
  }
};
int main(){
  volume v(2,3,5);
  v.getVolume();
  return 0;
}
```

* function overloading (constructor)

```
class Dimensions {
  private:
     int length;
     int breadth;
     int height;
  public:
     Dimensions(int len, int br, int h=0): length(len), breadth(br),
                                           height(h) {}
     void calcArea(){
        cout << length * breadth << endl;
     }
     void calcVolume(){
        cout << length * breadth * height << endl;
int main(){
     Dimensions rect(5,10);
     rect.calcArea();
     Dimensions box(5,10,4);
     box.calcVolume();
     return 0;
}
```

* member function overloading

```
class Dimensions {
  private:
     double length;
     double breadth;
     double height;
  public:
     void setDim(double l, double b) {
        len = l
        br = b
        h = 0
     }
     void setDim(double l, double b, double h) {
        len = l; br = b; he = h; }
     double calculate{
        if (height == 0) { return length * breadth}
        else {return length * breadth * height}
  };
```

```
int main(){
  Dimensions shape;
  Shape.setDim(2,4);
  cout << shape.calculate()
  Shape.setDim(2,4,5);
  cout << shape.calculate()
  return 0
}
```

* Dynamic memory Allocation

i) `int * numPtr = new int;`
`*numPtr = 5;`

```
int var = 34;
int *p;
p = &var
```

*p = 34

```
cout << " allocated int value " << *numPtr << endl;
delete numPtr; → deallocating → numPtr is now a dangling pointer
numPtr = nullptr;                (points to an invalid memory location)
```

ii) `int *arr = new int[size]`
`delete[] arr;`

```
int *ptr = arr  // ptr points to first element of the array
for(int i=0; i<size; i++){
    cout << *ptr << " ";
    ptr++;
}
```

* setw

```
cout << "n=" << setw(5) << endl
cout << "n=" << setw(5) << endl
```

o/p : n= 1234
78

* switch(expression) {
case const1:
action1;
break;
case const2:
actia 2;
break;
default:
action n;

* reference variable → second name for the variable (share same data space)

```
float total = 100;
float &sum = total
```

(same address)

```
cout << sum
```

100

```
cout << &sum
```

0x61ff08 → outputs the memory address where sum reference variable is stored in memory

* can be defined outside class using ::

* ENUM

i) enum char {a = 3, b = 7, c, d};

value of c is 8, d is 9 (default ⇒ increment)

ii) enum Color { Red, Green, Blue }

```
Color myColor = Green; // Declare a variable of type Color
if (myColor == Red) {
    cout << " color o red " << endl
else if ( myColor == Green) {
    cout << " color is green << endl
else if ( blue)
    cout
else { " invarid color " }
```

* # define Pi 3.14

* sizeof (variable)

int → 4
char → 1
float → 8
double → 8

* Default Arguments

```
void temp ( int = 10, float = 8.5);
int main() {
  temp()
}
```

i) void temp (int i, float f) // i=10, f=8.5 (default values)

ii) temp (6) ⇒ (i = 6 f = 8.5)

iii) temp (6, 2.5) ⇒ (i = 6, f = 2.5)

iv) ten (3.5) ⇒ (invalid) → missing argument must be last argument of list

```
int mul( int i, int j = 5, int k = 10);  // legal
int mul( int i = 0, int j, int k = 7);   // illegal
```

* Copy constructor:

```
Student (student &s) {
    rollno = s.rollno;
    strcpy (name, s.name);
    mark s = s.marks;
}
```

Inside main() {

Studat s5 = s4 (already paratermized initialized)

* Destructor

~Student();

* DYNAMIC ARRAY:

int *ptr = new int[10]

// allocates an array of 10 integers and returns a pointer to the first element in that array (which is used to initialize ptr)

For multidimensional arrays:

int* nums = new int[x][4][5]

↑ only first dimension can be a variable

delete [] ptr → indicates to compiler that the pointer addresses an array of elements

* MALLOC:

int * nums = (int*) malloc(5 * sizeof(int));

free (nums)

to call member function of class using ptr: ptr → disp();

Cls *ptr = new Cls // calls default constructor of cls

Cls *ptr = new cls // calls parameterized constructor of cls

// new returns a pointer to the object of the class.

* Public – Can be accessed from anywhere outside

Protected – are accessible only in a class derived from base class

Private – only accessible from within the class

* INHERITANCE

```
Class person {
    protected :
      ~~
    Public :
        void getdata(){
          cin >> ~~
        Void display(){
          cout <<~~
        }
};
Class employee : public person {
    Protected :
      ~~
      public :
          void getinfo(){
            person :: getdata();
            cin >> ~~
          }
          void dis(){
            person :: display();
            cout <<~~
          }
};
```

* Class teacher : Public person, Public employee

* POLYMORPHISM / VIRTUAL / OVERRIDE

```
#include <iostream>
class Shape {
public:
    virtual void draw() const {
        cout<<" drawing a shape"<<endl;
    }
};

Class Circle : public Shape {
public:
    void draw() const override {
        Cout <<"drawing a circle" <<end;
    }
};

Class Square : public Square {
public :
    void draw () const Override {
        Cout<< "drawing a square" <<endl;
    }
};

int main() {
    Shape * shape1 = new Circle();
    Shape * Shape2 = new Square();
    shape1 -> draw(); // calls Circle's draw method
    shape2 -> draw(); // calls Square's draw method
    delete Shape1;
    delete Shape2;
    return 0;
}
```

alternatively

```
Circle Shape1;
Square Shape2;
Shape1.draw();
Shape2.draw();
return 0
```

* ASCII

0 - NULL
32 - Space
48 - '0'
57 - '9'
65 - 'A'
90 - 'Z'
97 - 'a'
122 - 'z'

i) A+32 = 'a'

* FRIEND FUNCTION :

```
class book {
  private:
    int bno;
    char bname[20];
  Public:
    void getdata();
    friend void show(book);
};
void book :: getdata () {
  cin >> bno >> bname; }
  void show (book bk) {
  (out << bk.bno << bk.bname;
  }

main {
  book b;
  b. getdata();
  show(b);
```

```
Class ClassB; // forward declaration

class class A {
  Private:
    int A;
  public:
    void setAval(int val) {
      A=val
    }
    friend void FriendFunction(class A, classB);

Class classB {
  private:
    int B;
  public:
    void setBval
  friend void friendFunction (classA objA, ClassB objB) {
      Cout << objA. A << endl
      Cout << objB. B << endl
} }
```

```
int main() {
  ClassA objA;
  Class B objB;
  objA. setAval (5);
  objB. setBval (10);
  friendFunction
    (objA, objB);
  return 0;
  }
```

* FRIEND CLASS

```
class classB; // forward declaration

class classA {
    private:
        int numA;
        friend class classB;
    public:
        classA(){
            numA = 12;
        }
};

class classB {
    private:
        int numB;
    public:
        classB(){
            numB = 1;
        }
        int add() {
            classA objA;
            return objA.numA + numB;
        }
};

int main() {
    classB objectB;
    cout << "Sum: " << objectB.add();
    return 0;
}
```

→ Constructor

STANDARD TEMPLATE LIBRARY

1) ARRAY

```
#include <array>

array<int, 5> arr = {1,2,3,4,5};
int size = arr.size();
```

Element at 2nd index : arr.at(2) → outputs 3

arr.empty(); → bool 1 or 0 → not empty

FIRST ELEMENT: arr.front(); → first element

LAST ELEMENT: arr.back(); → last element

2) VECTOR

// dynamic allocate | doubles current size, copies current elements in new, dumps old

```
#include <vector>

vector <int> v;
```

- vec.begin(); → pointer to first element
- vec.end() → points to last element

v.shrink_to_fit(); → reduces capacity to fit its size & destroys elements beyond capacity

v.capacity(); → shows how much space is assigned

v.size(); → shows number of elements in vector currently

v.push_back(1); → adds element with value 1 to vector at the back

~~v.push_front(5); → adds element with value 5 to vector at the front~~

v.pop_back(); → removes last element in vector from the back

v.clear(); → empties vector

vector<int> vec(5, 2) → value assigned to all elements (0 by default); 5 → size (no. of elements)

vector <int> copy(vec);

v1.swap(v2);

~~vec.insert(vec.begin()+3, 100);~~ → value of inserted element

vec.insert(vec.begin()+3, 100); → inserts after 3rd element at position 3 (0-based)

3) DEQUE

```
#include <deque>

deque<int> d;

d.push_back(1);
d.push_front(2);   } → outputs 2 1

d.pop_back();
d.pop_front();     } → delete accordingly

d.erase(d.begin(), d.begin+2);  → will delete first 2 elements
        d.end()                 → full deque
```

3) LIST

```
//no random access provided  X l[i]
#include <list>

list<int> l;

l.push_back();   l.begin();   l.front();   l.empty();
l.push_front();  l.end();     l.back();    l.erase(l.begin());
l.pop_back();
l.pop_front();   l.size();
```

4) STACK

LAST IN FIRST OUT

```
#include <stack>

stack<string> s;
```

//new element is added at one end (top) and an element is removed from that end only

```
s.push("hello");   → adds at the back (top)
s.push("world");

s.top();  → outputs world

s.pop();  → deletes top element
```

5) QUEUE

FIRST IN FIRST OUT

//elements are inserted at the back and are deleted from the front

- #include <queue>

```
queue <string> q;
q.push ("hello");
q.push ("world");
```
→ queue will be {hello, world}

q.pop(); → removes from the front

```
void showq (queue<int> q)
 while (!q.empty()){
    cout<< q.front();
    q.pop();
  }
}
```

#include <algorithm>

max(a,b) swap min

reverse (s.begin, s.end);

sort(vec.begin(), vec.end()); → ascending order

sort (vec.begin(), vec.end(), greater<int>()); → descending order

6) PRIORITY QUEUE

priority_queue <int> maxq;

maxq.top(); → greatest element from the front

maxq.pop(); → element with greatest value is popped

priority_queue<int, vector<int>, greater<int>> minq;

7) MAP

→ key points to → value

first second

```
#include <map>
map<int, string> m;
m[1] = "hello";
m[12] = "world";
m[3] = "you";
m[4] = "such";
```

m.count(12); → 1

m.count(5); → 0

m.erase(1); → erases 1 erase by key

```
for (auto i:m) {
```

i.first → ascending order first element

i.second → corresponding string

auto it = m.find(3); → returns iterator to that particular element

EXCEPTION HANDLING

MULTIPLE CATCH STATEMENTS:

```
#include <iostream>
using namespace std;

int main () {

    double num, den, arr[4] = {0.0, 0.0, 0.0, 0.0};
    int index

    Cout << "Enter array index";
    Cin >> index;

    try {

        if ( index >= 4)
            throw "Error: Index out of bounds";

        Cin >> num >> den;

        cout << numerator;
    } catch
        if (den == 0)
            throw [illegible] 0;

        Cout << num / den << endl;
    } catch ( const char* str) {

        Cout << str << endl;

    } catch ( int i) {

        Cout << i << endl;

    } catch (...) {
        } Cout << "unexpected exception" << endl;
return 0;

}
```

STANDARD EXCEPTION :

```
#include <iostream>
#include <exception>
using namespace std;

int main() {
    int a=10, b=0, c=20;
    try {
        if (b == 0)
            throw runtime_error("divide by zero error");
        cout << c - (a/b) << endl;
    } catch (runtime_error &e) {
        cout << e.what() << endl;
    }
}
```

ii)

```
try {
    int* hugeArray = new int[1000000000000];
} catch (bad_alloc &e) {
    cout << e.what() << endl;
}
```

THROW FROM FUNCTION:

```
void exhandler(int* ptr, int num) throw(int*, int) {
    try {
        if (ptr == NULL)
            throw ptr;
        if (num == 0)
            throw num;
    } catch (...) {
        cout << "some exception was caught";
    }
}
int main() {
    exhandler(NULL, 7);
}
```

FILE HANDLING

```
#include <iostream>
#include <fstream>

int main(){
    ofstream fout;
    fout.open("fileHandler.txt");
    fout<<"Hello from program";
    fout.close();

    ifstream fin;
    fin.open("fileHandler.txt");
    char ch;
    while(1){
        fin.get(ch);
        if(fin.eof())
            break;
        cout<<ch;
    }
    fin.close();

    return 0;
}
```

```
string str2;
while(1){
    getline(fin, str2);
    if(fin.eof())
        break;
}
cout<<str2;
fin.close();
```

TEMPLATE CLASS - BUBBLE SORT

```
template<class T>   ⟶ alternatively: template<typename T>
void genericSort(T* arr, int size){
    for(int i=0; i<size-1; ++i){
        for(int j=0; j<size-1; ++j){
            if(arr[j] > arr[j+1]){
                T temp = arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
```