C++ Complete Course

Introduction V1

C++ → 1979 → Bjarne Stroustrup → extension of C

Features → fast program, more control over resources + Better memory management

Basic Structure V2

```
#include <iostream>
using namespace std; (1)
int main(){
(2) std::cout<<"Hello World";
return 0;
}
```


NOTE →

(1) Its used because it informs the system, that if you find something that's not declared in the current scope go and check std.

(2) std is a abbreviation used for standard. If we don't use (using namespace std;) we can use the standard namespace. In standard function of iostream we have functions like cin and cout. (:: → scope resolution operator)

Declaration of comment

V3

Syntax → // or /* */

VS code → • Select lines + Ctrl + / (forward slash)
• goto any line + ctrl + /

Variables

V4

→ Containers to hold data.

→ Based on scope it has 2 types:

- Local variables → declared inside the boacs of a function
- Global variables → declared outside of the function.

Rule of naming variables

- Varies from 1 to 255 character
- Start with letter or underscore
- Can contain numbers
- case sensative
- NO spaces or special character
- Should be reserved keyword

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Data type

→ It defines the type of data a variable can hold.

Datatypes in C/C++

- **Primary or Built-in**
  - Integer
  - Character
  - Boolean
  - Floating point
  - double floating point
  - void
  - wide character
- **Derived**
  - Function
  - Array
  - Pointer
  - Reference
- **User defined**
  - Class
  - ~~Pointer~~
  - Structure
  - Union
  - Enum
  - Typedef

2

Note →

int (-1, 0, 1, 7, 86)
Float (1.22, 3.7)
char ('a', 'd', 'A', '1')
double (1.21834) → Holds more precision than float
bool (0 or 1) → Either true or false.

Basic Input / output in C++ V5

In C++ sequence of bytes corresponding to input and output are commonly known as streams.

Input Stream → Dir^n of bytes takes place from input devices to memory devices.

Output Stream → Dir^n of bytes takes place from main memory to output devices.

<< (Insertion operator) → Send bytes to an output stream object.

4 [redacted] >> (Extraction operator) → Send bytes to an input stream object.

Data type size and range

Keyword : size in bytes : range
bool : 1 : 0 or 1
int (or signed int) : 2 : -32768 to 32767
Float : 4 : -3.4e38 to 3.4e38
double : 8 : -1.7e308 to 1.7e308
long : 4 : (same as long int)
unsigned int : 2 : 0 to 65535
unsigned char : 1 : 0 to 255
long int : 4 : -2,147,483,648 to 2147483647

V6

Including header files

#include <filename>

This method is used to include standard library header files. In this the pre processor searches for the files in the designated directory assigned by compiler.

#include "filename"

This method is used to include programmer defined header files. In this the pre processor searches for the files in the directory of file we are working currently.

Tuesday 6

Multicursor functionality

VS Code →
- Alt + shift + drag
- Alt + click

Pre and Post Increment / Decrement

var_name++
var_name--

} Post Increment / Decrement
After the execution of variable it alters its value by 1.

++var_name
--var_name

} Pre Increment / Decrement
It first alters the value of variable by 1 then executes the line.

V7

Scope resolution operator (::)

It can be used to call a global scope variable, if its ~~existence~~ hidden by any explicit declaration.

eg → `cout << ::var_name;`
`::var_name = 3;`

Decimal value in C++

If I input a decimal number in my program and ask my preprocessor about its data type then it will always denote it as double, as its more precise than float.

8 Thursday

So we need to specify earlier to setup its datatype.

```
float var_name = 34.4;
var_name = 34.4f;
var_name = 34.4F;
```

} They are declared as floating numbers. (PS → f/F will not get displayed)

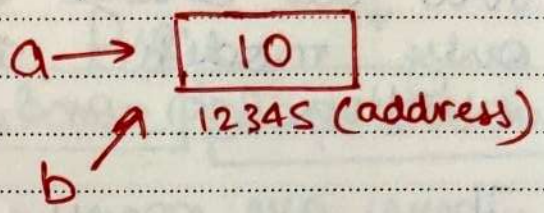
```
long double varname = 34.4;
var_name = 34.4L;
var_name = 34.4L;
```

These all changes matter in function overloading

Reference variable

Its a variable provided an alternate name.

eg → int a=10;
int &b=a;
or int& b=a;



Type casting

Its used to convert one data type to another data type.

Saturday 10

cin>>(float)var_name ;
cin >> float (var_name);

Constant function

V8

We can restrict the value of variable by creating a constant.

const float a=3.11;

11 2018 February
Sunday

January 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Manipulators in C++

These are helping functions that can modify the input / output stream. It doesn't mean that we change the value of variable, it only modifies the I/O stream using insertion (<<) and extraction (>>) operator.

There are many manipulators such as -

- <<endl; → moves the cursor to the next line
- setw (number) → Its inside the <iomanip> header file.

We can use it to right justify our output.

12 Monday

eg → int a=3, b=78, c=1233;

```
cout << a;
cout << b;
cout << c;

cout << setw(4) << a;
cout << setw(4) << b;
cout << setw(4) << c;
```

Output

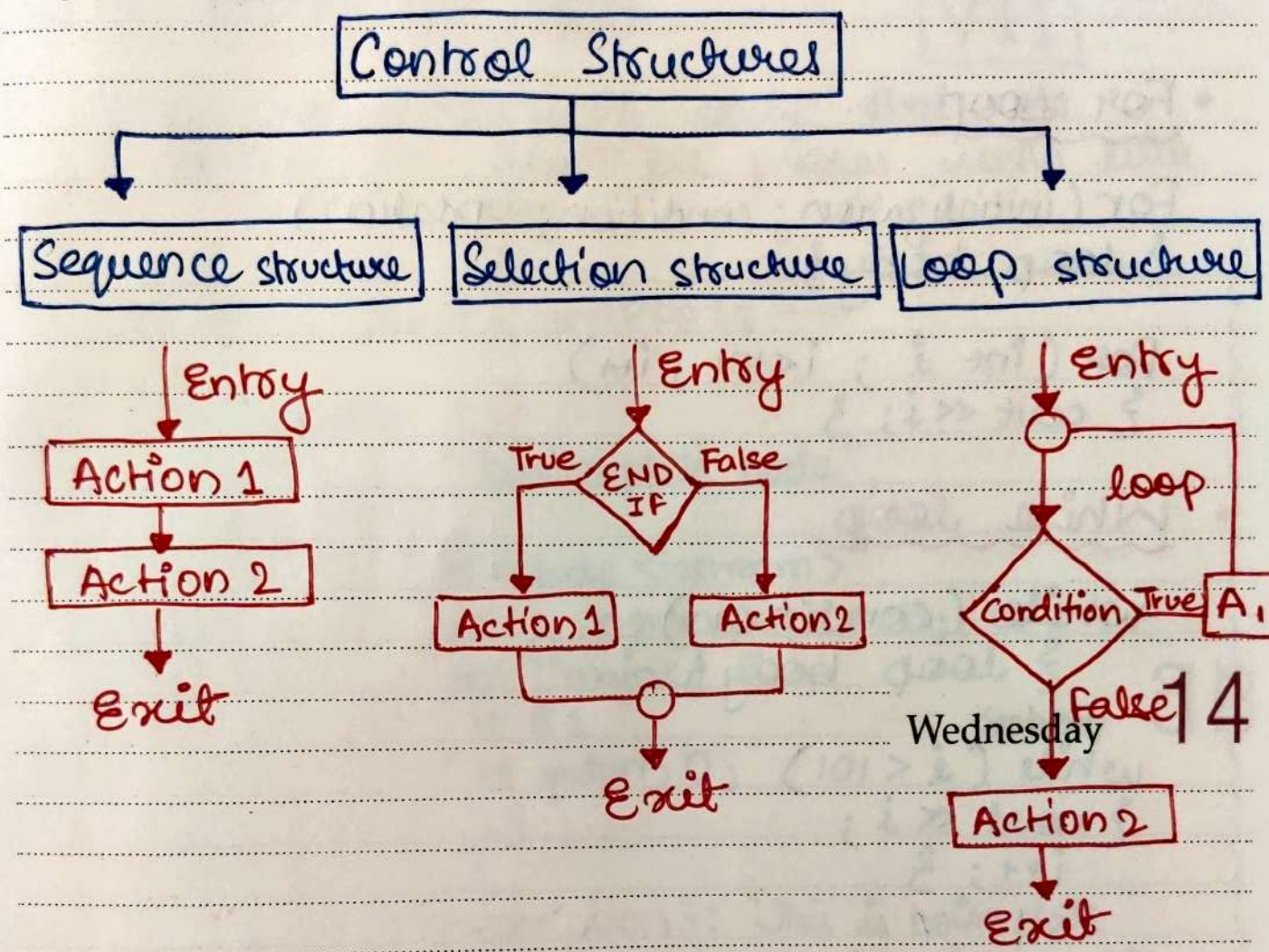
```
3
78
1233

   3
  78
1233
```

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

C++ Control Structures

V9



Switch Case Function

This can be used to create a menu driven program.

```
switch(var_name)
{case 'char': statement 1; break;
 case num: statement 2; break;
}
```

Switch case syntax.

15 2018 February
Thursday

January 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Loop Control Structure

V10

- **For loop**

```
for (initialization ; condition ; updation)
{ loop body }
```

```
for (int i ; i<101 ; i++)
{ cout << i; }
```

- **While loop**

```
while (condition)
  { loop body }
```

16 Friday

```
while ( i < 101)
{ cout << i ;
  i++ ; }
```

- **do while loop**

```
do
{ Statement # loop body }
while (condition);
```

```
do
{ cout << i ; i++ ; }
while ( i<101)
```

Boilerplate Code

V11

They are sections of code that are repeated in multiple places with little or no variation.

VS CODE USER SNIPPETS →

```
"boilerplate": {
"prefix": "boilerplate code",
"body": [
    "#include<iostream>",
    "using namespace std; \n",
    "int main() {",
    "\t $1",
    "\t return 0;",
    "}",
    ],
"description": "This is boilerplate"
    }
```

} → All the code is placed under the previous braces.

19 2018 February
Monday

January 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Break & Continue Statement

break; → Its used to exit the loop at a particular condition.

continue; → It skips the particular iteration and moves to the next one.

Pointer

V12

Its a data type, which can hold address of other data types.

20 Tuesday

```
int  a=3;
int* b=&a;
```

& → (Address of) operator

* → dereferencing (value at) operator.

```
cout << &a;    { 0x61FF08
cout << b;     { 0x61FF08
cout << *b;    { 3
```

```
int** c=&b;
```
→ We created a variable to store address of other pointer.

March 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

February 2018

Wednesday 21

V13

Arrays

user defined ~~data~~ datatype – store similar collection of data.

datatype var_name[x] → Array declaration

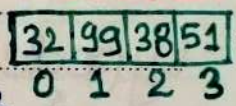
int marks[4] = {32, 99, 38, 51}

↳ Compiler is intelligent enough if we don't declare the parameter it still fine.

int a=3;

&a → we use this to get address of a.

marks → address of first block



Thursday 22

&marks

↳ This is a wrong approach for getting address

int* p = marks; → Pointer p to store address of first block of array.

p++;

↳ Address of index 1.

∴ If we dereference the pointer we get

*p = 32 | *(p+1) = 99 | *(p+2) = 38 | *(p+3) = 51

23 2018 February
Friday

January 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Pointer Arithmetic

address new = address old + i * size of (datatype)
(P+i) (P)

Structure

V14

- user defined datatype
- combines different datatypes
- declared outside main()

24 Saturday

```
typedef struct employee
{
    int eId;      // 4 bytes   } 9
    char favchar; // 1 byte    } Bytes
    float salary; // 4 bytes   }

}; ep;

main()
{ struct employee aryan;
//ep aryan;

aryan.eId = 1;
aryan.favchar = 'c';
aryan.salary = 12000000;
cout << aryan.eId; }
```

March 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

February 2018

Sunday 25

Union

They are like structures but provide better memory management.

- Used if we need just one characteristic.
- declared outside main().

```
union money
{ int rice;   //4 bytes
  char car;   //1 bytes
  float pounds; //4 bytes
};
```

} Would take max 4 bytes of memory.

Monday 26

enum

Its a datatype that provides indexing.

- Increases readability of code
- declared inside main().

```
main()
{ enum Meal {breakfast, lunch, dinner};
  Meal m1 = lunch;
  cout << (m1 == 2);
  cout << breakfast;
  cout << lunch;
  cout << dinner;
```

```
0 // False
0   #OUTPUT
1
2
```

27 2018 February
Tuesday

January 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

V15

Return

Return 0 → denotes that the program executed succesfully.

Return 1 → denotes that the program executed but we even created a error we are aware of.

eg → exceeding the size of datatype.

* In some cases of execution we even get random garbage values.

28 Wednesday

Functions

```
int sum (int a ,int b)
{ int c=a+b;
  return c; }
```

Function prototyping

```
int sum(inta, int b);
int sum( int, int );
```

It informs the compiler that which functions are going to come further.

```
int sum (inta , b);
```

↳ Not acceptable

Actual Parameter → used in the main functions

Formal Parameters → Uses the value of actual parameter to do operations.

Call By Value & Call By reference

V16

```
void swappointer (int* a, int* b)
{ int temp = *a;
  *a = *b;
  *b = temp; }
swappointer (&a, &b);
```



```
void swapreference (int& a, int& b)
{ int temp = a;
  a = b;
  b = temp; }

swapreference (x, y);
```

V17

Inline functions

```
inline int product (int a, int b) {
return a*b; }
```

It just copies the same lines of code in the function, wherever this function has been called.

Note → Don't make large sized functions as inline functions, either you won't get the advantage that we are looking for i.e reduced in time of execution.

Static declaration

```
Static int c=0
```

↳ This would just run once in the loop continuation.

// Not recommended for inline functions as the value is retained.

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

March 2018

Monday 5

Default argument

```
float money(int initialamt, float factor=1.25)
```

Constant argument

```
int strlen(const char * p)
// Don't change the value of p or it might throw back error.
```

Recursion

V18

Tuesday 6

```
int factorial
{ if (n<=1)
    return 1;

return n * factorial(n-1); }
```

Its the calling of function inside itself.

Function overloading

V19

```
void function()
void function(int a)
void function(int a, float b)
```

Its the calling of different function with same name.

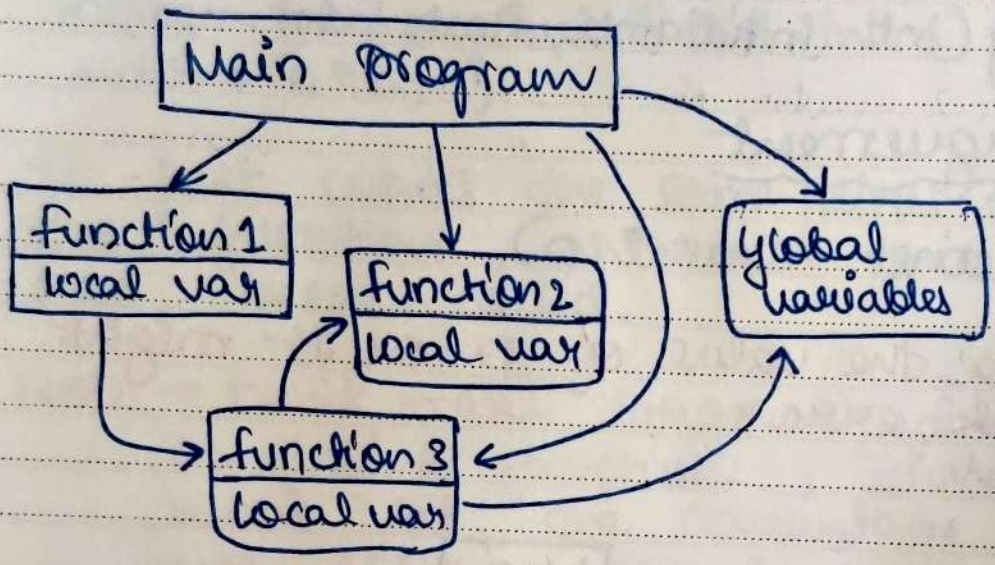
7 2018 March
Wednesday

February 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

Procederal oriented programming Vs OOPS

V20



8 Thursday

Basic Concept of OOPS

- Data Abstraction → wrapping data & function
- Data Encapsulation → hiding data in single entity.
- Data Inheritance → properties inherited to other
- Polymorphism → ability to take multiple forms.
- Dynamic Binding → code which will extract is unknown until run.
- Message passing → Object, message (information)

April 2018 | March 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Friday 9

Benefits

- Reusability
- Data hiding
- Multiple object without interference
- software complexity can easily be managed.

Classes : Public & private

V21

```
class Employee
{ private:
    int a, b, c;
  public:
    int d,e;
    void setdata(int a, int b, int c);
    void getdata()
    { cout << a << b << c << d << e; }

void Employee :: setdata(int a1, int b1, int c1)
  { a=a1;
    b=b1;
    c=c1; }

int main() {
  Employee aryan;
  aryan.a = 134;
  aryan.d = 34;
  aryan.e = 89;
  aryan.setdata(1,2,4);
  aryan.getdata();
  return 0; }
```

would cause error as its a private member.

Saturday 10

11 2018 March
Sunday

February 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

V22

Declaring of Classes

C++ >> initially called >> C with classes by Bjarne Stroustrup

classes → extension of C (structures)

```
class Employee {
} Aryan, Rohan ;
```

Declaring class object named Aryan & Rohan.

aryan.salary = 8 , makes no sense if salary is in private list. we need to declare it by method.

12 Monday

// nesting of member function

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

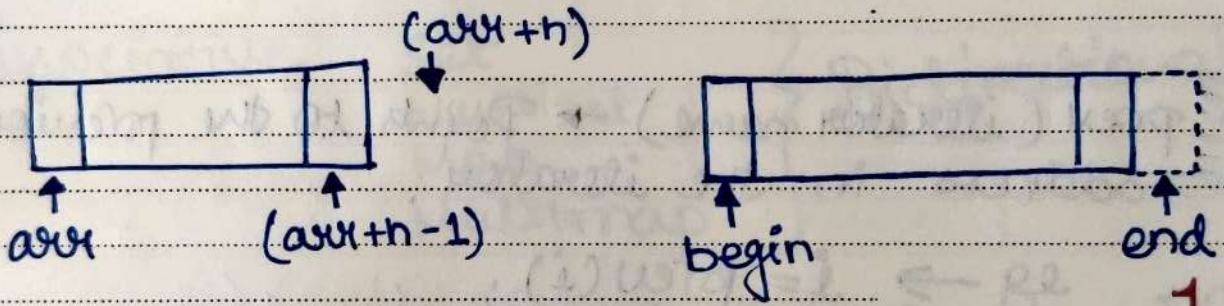
March 2018

Tuesday 13

C++ STL

Containers

- Simple : Pair, Vector, forward_list, list
- Containers Adapter : Stack, Queue, Priority_queue
- Associative : set, map, unordered_set, unordered_map



Wednesday 14

Iterator → gives us the address of an element in a container.

CODE

```
main()
{vector<int> V = {10,20,30,40,50};

vector<int>::iterator i = v.begin();
// auto i = v.begin()
cout << (*i);
}
```

Inbuilt functions for iterators →

- next (iterator name) → Points to the next address in the iterator.

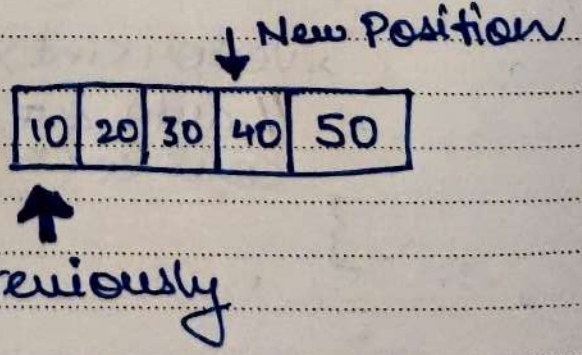
eg → `i = next(i);`
`cout << *i;`

- prev (iterator name) → Points to the previous address in the iterator.

eg → `i = prev(i);`
`cout << *i;`

- advance (iterator_name, skips_no) → It skips the position of the iterator n steps ahead.

eg → `advance(i, 3);`



Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

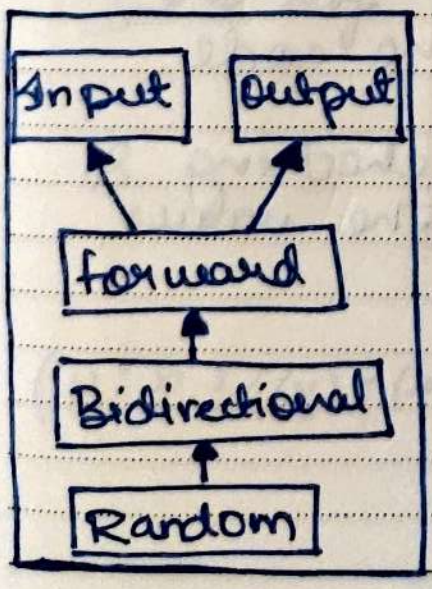
* Iterators in STL

Simple
- forward_list → forward
- list → Bidirectional
- vector → Random

Associative
- Set, Multiset, Map, Multimap } Bidirectional
- unordered_set, unordered_map } forward

Sunday 18

Adapters



- Queue
- Stack
- Priority queue

} Do not have iterators

February 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

* Templates → Write it once and use with any data type.

• Similar like macros it initializes at start.

eg →
```
template <typename T>
T mymax (T x, T y)
{ return (x>y)? x:y; }

main()
{ cout << mymax <int> (3,7);
cout << mymax <char> ('c','g');
return 0;
}
```

20 Tuesday

* Macros → It just searches for the lines and replaces the code.

Drawback → Doesn't do type checking & just replaces the values.

eg →
```
#define mymax ((x)>(y))? x:y)
```

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

March 2018

Wednesday 21

* Function Templates →

eg →
```
template <typename T>
T arrmax (T arr[], int n)
{
    T res = arr[0];
    for (int i=1 ; i<n ; i++)
        if (arr[i] > res)
            res = arr[i];
    return res;
}

//function to return the largest element

main()
{
  int arr[] = {10,40,3};

  cout << arrmax<int> (arr1, 3);

}
```

Thursday 22

23 2018 March
Friday

February 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

* Class Templates →

eg →
```
template <typename T>
struct pair {
  T x, y;
  Pair(T i, T j) { x=i; y=j; }

  T getfirst()
  { return x; }

  T getsecond()
  { return y; }
  };
```

24 Saturday

```
int main()
{
  pair<int> P1(10,20);
  cout << P1.getfirst() << P1.getsecond();
  return 0;
}
```

Simple containers

#Pair → It is used to combine 2 values together that are different in datatype.

eg → x, y coordinates
item name and price

Code

```
#include <utility>
#include <iostream>
using namespace std;
int main()
{ pair <int, int> p1(10,20);
  pair <int, string> p2(10, "GFG");
  cout << p1.first << " " << p1.second << endl;
  cout << p2.first << " " << p2.second << endl;
  return 0;
}
```

February 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

* Comparision of pairs

```
int main()
{ pair<int, int> p1(1,12), p2(9,12);

    cout << (p1 == p2);
    cout << (p1 != p2);

// In this we compare the first and second
// element if both are same then returns
28 Wednesday // 1 else 0.

    cout << (p1 > p2);
    cout << (p1 < p2);
    return 0;
}
```

// This only compares the first element of the pair and the second doesn't matter.

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

March 2018

Thursday 29

Sequenced Containers

Vectors → Vectors are same as dynamic array with capability to resize itself.

eg → vector<int> v;

```
v.push_back(10);
v.push_back(15);
```

v[i] → We can use the index similarly as in arrays, but it doesn't do index out of bound checking.

Friday 30

v.at(i) → It does index out of bound checking.

```
for (int i=0; i<v.size(); i++)
    cout << v[i];
```

```
for (int x : v)
    cout << x;
```

↳ This won't cause any change to main values.

```
for (int &x : v)
    x = 6;
```

↳ This would change the real value

```
for (auto it = v.begin(); it != v.end(); it++)
    cout << *it
```

↳ called by using iterators

February 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

- ```
  int n=3, x=10;
  vector<int> v(n,x);
  ```

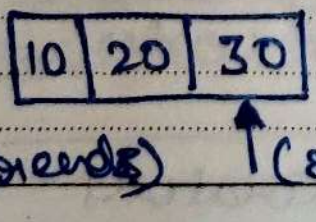
↳ It creates a vector of size 3 where all elements are 10.

- ```
  arr[] = {10,20,30};
  int n = sizeof(arr) / sizeof(arr[0]);
  vector<int> v(arr, arr+n)
  ```

↳ This creates a vector with all the elements of array, starting from arr and ends before (arr+n).

- ```
  for (auto it = v.rbegin(); it != rend(); it++)
  cout << *it;
  ```

↳ This traverses the vector from the back.


  ```

Various functions in vector

- pop_back() →

`v.pop_back();` → This removes the last element of the vector.

- front() →

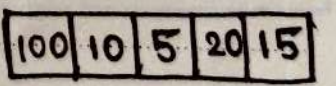
`v.front()` → This indicates the first element of vector.

- back() →

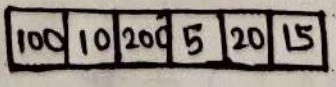
`cout << v.back();` → This would shows us the last element of vector.

- insert() →

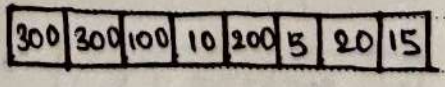
```
vector<int> v[10,5,20,15];
auto it = v.insert(v.begin(),100);
```



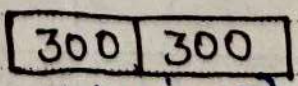

```
v.insert(v.begin()+2, 200);
```




```
v.insert(v.begin(), 2, 300);
```




```
vector<int> v2;
v2.insert(v2.begin(), v.begin, v.begin+2);
```



↳ would not include this position.

March 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

• erase() →

`v.erase(v.begin())` → Removes the first element of array.

`v.erase(v.begin(), v.end()-1)`

↳ Removes all the elements in the vector.

• clear() →

`v.clear()` → It clears all the elements of the vector.

• empty() →

```
if(v.empty() == true)
  cout << "AK";
```

→ It checks whether the vector is empty or not.

• resize() →

`v.resize(5)` → Resizes the array to size 5

`v.resize(8,100)` → Resizes the array to size 8 and keep 100 at the new spaces.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

* Time complexity of vector functions

push_back()
pop_back()

front()
back()
empty()

begin(), rbegin(), crbegin(), cbegin()
end(), rend(), crend(), cend()
size

} O(1)

6 Friday

While using user defined function, this might increase more time to execute the program.

insert()
erase()
resize()

} O(n)

Sequenced Containers

forward_list → Implements singly linked list (implementation)

forward_list <int> l = {10, 15, 20}; 10 → 15 → 20

l.push_front(5); 5 → 10 → 15 → 20

l.push_front(3);
l.pop_front(); 5 → 10 → 15 → 20

8 Sunday

`l.assign({10,20,30,10});`
↳ Assigns a new set of values to the forward list.

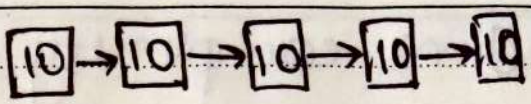
`l.remove(10);` → Removes all the copies of 10 in the forward_list.

20 → 30

```
forward_list <int> l2;
l2.assign(l.begin(), l.end());
```
↳ Makes another list l2, with all the values of l.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

- `l.assign(5,10)`



↳ Assign 5 places in the forward list and each of value 10.

- `forward_list<int> l1 = {15,20,30};`
  `auto it = l1.insert_after(l1.begin(), 10);` 15,10,20,30
  `it = l1.insert_after(it, {2,3,5});` 15,10,2,3,5,20,30
  `it = l1.emplace_after(it, 40);` 15,10,2,3,5,40,20,30
  `it = l1.erase_after(it);` 15,10,2,3,5,40,30

Tuesday 10

- `clear()` → clears the list
- `empty()` → Checks if the list is empty or not.
- `reverse()` → [`l.reverse()` - Reverses the list]
- `merge()` →

```
forward_list l1 = {10,20,30};
forward_list l2 = {5,15};
l1.merge(l2);
```

→ [5,10,15,20,30] l1, [] l2

Merge functions, inserts the element in sorted order.

- `sort()` → It sorts the list

11 2018 April
Wednesday

March 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

* Time complexity of forward_list

insert_after()
erase_after()
assign()
} O(1) for one item and O(m) for m items.

push_front()
pop_front()
} O(1)

reverse()
remove()
} O(n)

12 Thursday

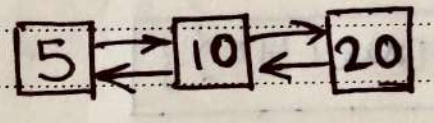
sort() } O(n log n)

Sequenced Containers

list → It's a doubly linked list

```
list<int> l;

l.push_back(10);
l.push_back(20);
l.push_front(5);
```



- pop_front() → Removes the front element
- pop_back() → Removes the last element

Saturday 14

- list<int> l = {10,20,30}

```
auto it = l.begin();
it++;
it = l.insert(it,15);
l.insert(it,2,7);
```

10, 20, 30
10, 15, 20, 30
10, 7, 7, 15, 20, 30

Insert function inserts the element before the iterator.

- size() → It gives the size of the list.

15 2018 April
Sunday

March 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

- `it = l.erase(it);`
  - ↳ It removes the element, this iterator is pointing toward and then the iterator points to the element after the removed element.
- `l.remove(40);`
  - ↳ Searches for all occurrances of 40 in the list and removes it.
- `l1.merge(l2);`
  - ↳ works similarly as in forward list.

16 Monday

★ List usually is a doubly linked list so we can say that it has a head and tail. So that we can do many operations in less time.

- `l.unique();` → Removes all consecutive duplicates
- sort() → Sorts the list
- reverse() → Reverses the list

* Time complexity of list functions

front()
back()
size()
begin()
end()
erase(it)
} O(1)

push_front()
pop_front()
push_back()
pop_back()
} O(1)

reverse()
unique()
remove()
} O(n)

sort() } O(n log n)

19 Thursday 2018 April

March 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Sequenced Containers

Dequeue → It provides functionality of both stack and queue.

- In this we can insert & delete from both the ends.
- In Dequeue, over C++ we have some extra benefits like we have random access, and allows us to do arbitarary no of insertions in O(1) time.

20 Friday

- dequeue<int> dq={10,20,30};
  dq.push_front(5); 5,10,20,30
  dq.push_back(50); 5,10,20,30,50

- dequeue<int> dq = {10,15,30,5,12}
  auto it = dq.begin();
  it++;

dq.insert(it,20); [10,20,15,30,5,12]
dq.pop_front(); [20,15,30,5,12]
dq.pop_back(); [20,15,30,5]

cout<< dq.size();

May 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

April 2018

Saturday 21

* Time complexity of Dequeue functions

push_back
push_front
pop_front
pop_back
} O(1)

insert()
erase()
} O(n)

Sunday 22

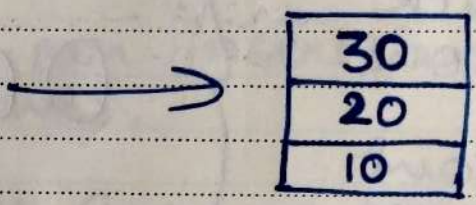
March 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Container Adapter

Stack → LIFO or FILO {Stack of plates}

```
stack<int> s;
s.push(10);
s.push(20);
s.push(30);
```



- s.size() → Denotes the number of elements in the stack.
- s.top() → Shows the last entered element of the stack.

- s.pop() → Removes the last entered element or the first element of the stack.
- s.empty() → Check whether the stack is empty or not.

May 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

April 2018

Wednesday 25

* Time Complexity of various function in stacks.

Stack can be implemented on any underlying container, that provides the following operations -

- back()
- size()
- empty()
- push_back()
- pop_back()

push()
pop()
top()
size()
empty()
} O(1)

Thursday 26

March 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

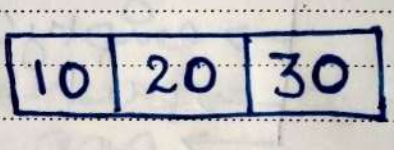
Container Adapter

Queue → FIFO

The person came first would be served first.

- #include <queue>

- queue<int> q;
  q.push(10);
  q.push(20);
  q.push(30);



28 Saturday

- q.front() → It denotes to the first element which is going pop next.
- q.back() → It denotes the last element which is going to pop out.
- q.pop(); → It removes the element which is entered first.
- q.size() → No of elements in Queue.
- q.empty → To check whether the Queue is empty or filled.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Sunday 29

* Time Complexities of various functions in Queue.

Queue can be implemented on any underlying container, that provides the following functions.

- empty()
- size()
- front()
- back()
- push_back()
- push_front()

Monday 30

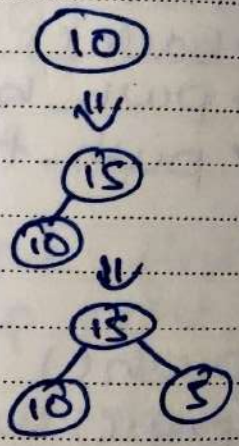
- push()
- pop()
- front()
- back()
- empty()
- size()

} O(1)

☑ ~~Priority Queue~~ Container Adapter

Priority Queue →

- Always implemented using heap data structure.
- Here it always uses Max heap, i.e always gives the current max element.
- #include <queue>
- priority_queue<int> pq;
  pq.push(10);
  pq.push(15);
  pq.push(5);



2 Wednesday

- pq.size(); → No of elements in P.queue
- pq.top(); → Max^m element {current max}
- pq.empty(); → Check if empty or not
- pq.pop(); → Removes the top element.

- `priority_queue<int, vector<int>, greater<int>> pq;`

↳ This creates a priority queue of smallest element at top. i.e min heap.

- `int arr[] = {10, 5, 15};`
  `priority_queue<int> pq(arr, arr+3);`

↳ This creates a priority queue using the array elements.

▲ Better to use this approach to input element, rather than pushing one by one. As has time complexity of O(n).

* Time complexity of priority Queue functions.

push() → log(n)
pop()
empty() } O(1)
size()
top()

5 2018 May
Saturday

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Associative Container

Set → They are special containers that store unique elements in a particular order, i.e increasing by default.

- #include <set>

```
set<int> s;
s.insert(10);
s.insert(5);
s.insert(20);

for(auto x:s)
    cout<<x<<" ";
```

5 10 20

6 Sunday

- set<int, greater<int>> s;
  ↳ Decreasing order to store elements
  i.e 20 10 5

- s.begin()
- s.end()
- s.rbegin()
- s.rend()

- s.erase(5) → Removes element 5
- s.clear() → clears the set.

- auto it = s.find(3);
  if (it == s.end())
  cout << "Not found";
  else
  cout << "found";

  ↳ If the elements gets found, then it places the position of the iterator. But if its not there it would still point to the last iteration.

Multi Set → In these we can store multiple instances of the same element, in increasing order.

Tuesday 8

- multiset<int> ms;

Note → Set is based upon self-balancing binary tree, in particular Red-Black tree.

9 2018 May
Wednesday

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Time complexities of functions in Set and Multiset.

begin(), end()
rbegin(), rend()
cbegin(), cend()
crbegin(), crend()
size(), empty()
} O(1)

insert(), find()
count(), lower_bound()
upper_bound(), erase(value)
} O(logn)

10 Thursday

erase(it) → Amortized O(1)

↳ If in this Red-Black tree, we have n delete operations and we have address of each element, then it can do operations in O(1).

Associative Container

Map → It is used to store a key & value pair, in ordered manner.

- It also utilizes Red-Black tree
- #include <map>
- map<int, int> m;
  m.insert({10, 200}) or m[10] = 200;
  } Inserts the key & value pair

Saturday 12

- for (auto &x : m)
  cout << x.first << " " << x.second
  ↳ key ↳ value.
  ↳ The order would be in the increasing order of the key values.

- m.size() → Tells us the no of pairs
- cout << m[20];
  ↳ If we tried to call an element, that's not present. Then it would add that pair to the map, and keep its value pair as 0.

13 2018 May
Sunday

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

- m.at(10) = 300;
  ↳ This ensures if the following key is present or not, if not present it throws an exception error.
- m.clear() → clears the map list
- m.find(5) → checks if the key is present
- map<int, string, great<int>> m;
  ↳ This would save the keys in decreasing orders.

14 Monday

- m.erase(5) → This would erase the key from the map.
  ↳ m.erase(m.find(5))
  m.erase(m.find(2), m.end()

In this variation, we can even pass iterator as an argument.

MultiMap → In this we can create multiple key value pairs with the same key value.

`mp[10] = 20;` → we don't have this in the multimap.

- `#include <map>`
- `multimap <int, int> mp;`
- `mp.count(10)` → count the no of times, that key appears in map.

Time Complexities of function in map and Multimap

begin(), end() rbegin(), rend() cbegin(), cend() crbegin(), crend() empty(), size()	O(1)
count() find() erase(key) insert []	O(logn)

17 2018 May
Thursday

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Associative containers

#unordered set → This doesn't follow Red-Black tree, it works on hashing.

- It saves the element in any order.
- #include <unordered_set>
- unordered_set<int> us;
- All functions are same as sets.

18 Friday #Time complexity of various functions in unordered sets.

begin(), end()
rbegin(), rend() } O(1)

insert(), erase(val)
erase(it), find()
count } O(1) on average

size(), empty() } O(1)

Associative containers.

Unordered Map →

- Used to store key, value pair
- uses hashing
- No order of keys.

- #include <unordered_map>
- unordered_map <string, int> um;
- if (um.find("idle") != m.end())
  cout << "found!";
  else
  cout << "Not found";

↳ Checks if the element is present or not. If its not present, it would point the iterator to end() position

Time Complexities of various function in unordered map

begin(), end(), size(), empty()	O(1) at worst case #always
count(), find(), erase(key), insert(), [], at	O(1) on average

21 2018 May
Monday

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

STL Algorithms

1. Non-mutating Algorith → These algorithm do not change the order of elements. They use iterators for particular operations.

eg → max_element()
min_element()
accumulate()
find()
binary_search()
rotate()

lower_bound()
upper_bound()
rotate()
fill()
is_permutation()
rand()

```
#include <algorithm>
```

22 Tuesday

2. Mutating Algorithms → These algorithms are modifying algorithms that are designed to work on the container elements and performs operations like shuffle, rotation, changing the order and more.

eg → sort()
reverse()
next_permutation()
prev_permutation()
make_heap()
merge()

* Various STL Algorithm functions

- find() → It is used to find an element in a vector or array.
- It is present inside std library.
- Its better to use STL::find() instead of find().

```
vector<int>v={5,10,7,10,20}
auto it =find(v.begin(), v.end(), 10);

if (it == v.end())
   cout<< "Not found";
   else
   cout<<" found at " <<(it-v.begin);
```



```
int arr[] = {5,10,12,8,7,3};
int  *ptr = find (arr, arr+6, 7);

if (ptr == (arr+6))
   cout<<"Not found";
else
   cout <<"found at" <<(ptr-arr);
```

25 2018 May
Friday

April 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

- lower_bound() → In this we put a value as an argument, and it searches for the element just greater than or equal to the entered element.

* Its needs to be sorted, to get proper result. As it uses binary search.

* Returns an iterator having address of element greater than equal to the given value in sorted range.

- vector<int> v = {10, 20, 30, 40};
  auto it = lower_bound(v.begin(), v.end(), 20)
  cout << *it << endl;
  it = lower_bound(v.begin(), v.end(), 25);
  cout << *it << endl;

26 Saturday

20 ←

30 ←

- int arr[] = {10, 20, 30, 40}
  auto it = lower_bound(arr, arr+6, 50);

↳ As we don't have any bigger element in set, it would point to (arr+6).

- Time Complexity: O(log n)
  → for random access containers.

- upper_bound() → In this it returns an iterator pointing to the first element in the range [first, last) that is greater than value or last if no such element is found.

- This even needs to be sorted.

- vector<int> v = {10, 20, 20, 20, 30, 40}
  auto it = upper_bound(v.begin(), v.end(), 20);
  cout << (it - v.begin()) << endl;
  ↳ 4 {As the index}

- if (it != v.begin() && (*(it-1) == x))
  cout << "found" << (it - v.begin());
  else
  cout << "Not found";

Monday 28

- Time Complexities : O(log n)
  → When we have random access. n is the total no of elements in the range.

- is_permutation() → • It is used to check that the given 2 set of containers have the same permutations.

- This ensures that all the elements in one container is present in another.

- The elements and the count of elements should be same.

- vector<int> v1 = {10, 30, 20, 5, 20};
  vector<int> v2 = {20, 10, 5, 30, 5};

```
if(is_permutation(v1.begin(), v1.end(), v2.begin);
        cout << "yes";
else
        cout << "NO";
```

30 Wednesday

↳ This would print NO, as the count is uneven.

- We can even use it for string, but for limited size string it would be a overkill.
  But incase of unlimited string, we can use it to compare.

- max_element() and min_element() →
- Does as the name, finds the max and min element inside the container.

- vector <int> v = {10,5,30,40,90,8}

```
auto it1 = max_element (v.begin(), v.end());
auto it2 = min_element (v.begin(), v.end());
cout << *it1 << *it2 ;
          ↳90    ↳5
```

- struct Point {int x, int y;
  Point (int i, int j)
  {x=i; y=j}
  };

```
bool cmp(Point p1, Point p2)
    {return p1.x < p2.x}

int main() {
vector<Point> v = {{5,4}, {2,300}, {90,10}};
auto it = max_element (v.begin(), v.end(), mycmp );
cout <<((*it).x) <<" " <<((*it).y) <<endl;
return 0 }; ⟶ 90 , 10
```

- Time Complexity : $\Theta(n)$

1 2018 June
Friday

May 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

- count() →
It counts the member of occurances of a particular element in a container.

- vector<int> v = {30, 20, 5, 10, 6, 10, 10}

cout << count(v.begin(), v.end(), 10);

↳ 3

2 Saturday

- str s = "aryankashyap";
cout << count(s.begin(), s.end(), 'a');

↳ 4

- use this count for array, vector, list, string but for other containers use their respective container count.

- binary_search() → It is a sorting algorithm which searches a sorted array by repeatedly dividing the search interval in half.

- vector<int> v = {10, 20, 30, 40, 50};

```
int x=25;
if (binary_search(v.begin(), v.end(), x)==true)
    cout << "Found";
else
    cout << "Not Found";
```

- We can even use user defined functions to compare / search various classes.

Monday 4

- Time Complexity :

If random access → O(log n)
If not random access → O(n)

5 2018 June

Tuesday

May 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

- fill() → This function assigns the value to the given range.

```
vector<int> v = {10, 20, 30, 40};

fill (v.begin(), v.end(), 5);
for (int x : v)
cout << x << " ";
```

→ 5 5 5 5

6 Wednesday

```
string s = "geeks";
fill (s.begin()+1, s.end(), 'A');
cout << s;
```

→ gAAAA

- rotate() → It rotates the order of elements in the range [first, last] in such a way, that the element pointed by middle becomes the first new element.

- vector<int> v = {10, 20, 30, 40, 50, 60};
  rotate(v.begin(), v.begin()+2, v.end());
  for(int x : v)
  cout << x << " ";

↳ 10 20 30 40 50 60
first, Middle, last

So the output would be :-
30 40 50 60 10 20

- int arr[] = {10, 20, 30, 40, 50, 60}
  rotate(arr, arr+2, arr+6)

- Time complexity → $\Theta(n)$

- accumulate() → By default, this function returns the sum of all values, lying in the range between [first, last) with the variable sum.

- vector<int> v = {10, 20, 30};
  init_res = 0;
  cout << accumulate(v.begin(), v.end(), init_res);
  ↳ Sum all vector elements

- int init_res = 100;
  cout << accumulate(v.begin(), v.end(), init_res, minus<int>)
  ↳ This would subtract all the vector elements from the predefined variable.

10 Sunday

- int myfun(int x, int y)
  { return x * y; }

  int main() {
  init_res = 1;
  int arr[] = {10, 20, 30};

  cout << accumulate(arr, arr+3, init_res, myfun);
  return 0; }
  ↳ Does product of all elements

- Time complexity → O(n)

- rand() → It is used to generate random numbers.

- #include <cstdlib>

- srand(time(NULL));
  for (int i=0; i<5; i++)
  cout << rand();
  → If we don't write this, it would display the same random no 5 times.

- cout << rand()%100;
  ↳ This would display numbers 0 to 99.

- int low = 10, high = 100;
  int range = high - low + 1;

Tuesday 12

```
for (int i=0; i<n; i++)
    cout << (rand()%range) + low << " ";
```

13 2018 June
Wednesday

May 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

- Sort → It is used to sort any container with random access, such as Array, vector, deque

- #include <algorithm>
- sort (arr, arr+n);
- sort (arr, arr+n, greater<int>);

- sort (v.begin(), v.end())

14 Thursday

- struct Point
  {int x, y;}

bool cmp (Point p1, Point p2)
{return (p1.x < p2.x)}

```
int main() {
  Point arr[] = [ {3,10}, {2,8}, {5,4} ];
  sort (arr, arr+n, cmp);
  for (auto i : arr)
    cout << i.x << " " << i.y ;
}
```

- make_heap() → It is used to transform a sequence into heap. A heap is a data structure which points to highest element and make it access in O(1) time.

- vector<int> v = {15, 6, 7, 12, 30}
  make_heap(v.begin(), v.end());
  cout << v.front() << endl;
  ↳ 30

- make_heap(v.begin(), v.end(), greater<int>());
  ↳ Min element at top

- pop_heap(v.begin(), v.end(), greater<int>());
  ↳ This would not change the container, just would place the element at top to the last and would ignore it, so that the next element would be at top.

- sort_heap(v.begin(), v.end(), greater<int>());
  ↳ 30 15 12 7 6

- push_heap(v.begin(), v.end(), greater<int>());

17 2018 June
Sunday

May 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

- merge() → This function merges 2 containers and passes them to a third container.

- ```
  vector <int> v1 = {10, 20, 40};
  vector <int> v2 = {5, 15, 30};
  vector <int> v3(6);

  merge (v1.begin(), v1.end(), v2.begin(),
         v2.end, v3.begin());
  ```

  ↳ 5 10 15 20 30 40

18 Monday

- ```
  int ar1[] = {10, 20, 30};
  int ar2[] = {5, 15, 40, 80};
  int ar3[7];

  merge (ar1, ar1+3, ar2, ar2+4, ar3);

  for (int x : ar3)
      cout << x << " ";
  ```

- Its basically preferred to sort the merge, as if the containers are not sorted it would give diff results.

- Time Complexity → O(m + n)

  m & n are elements in the containers.

- next_permutation() → It is used to rearrange the elements, in the range into the next lexicographical greater permutation.

- vector<int> v = {1,2,5,4,3}
  next_permutation(v.begin(), v.end())
  for(int x : v)
    cout << x;
  ↳ 1 3 2 4 5

- prev_permutation() → It is used to store the previous order in the lexicographical permutation. Wednesday 20

- vector<int> v = {5,4,1,2,3};
  prev_permutation(v.begin(), v.end())
  for(int x : v)
  cout << "x" << " ";
  ↳ 5 3 4 2 1

21 2018 June
Thursday

May 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

- reverse() → It is used to reverse the order of elements in the given range.

- vector<int> v = {10, 20, 30, 40, 50};
  reverse(v.begin()+1, v.end());
  for(int x : v)
  cout << x << " ";

  ↳ 10 50 40 30 20

22 Friday

- string s = "geeks";
  reverse(s.begin(), s.end());
  cout << s;

  ↳ skeeg

- We can reverse any container that has bidirectional iterator.
  (forward_list won't work)

- Time Complexity → $\Theta(n)$

Miscellaneous

- String class →

- string str = "aryan";
  cout << str.length() << endl;

  str = str + "kashyap";
  cout << str; ⟶ aryankashyap

  cout << str.substr(1,3) << endl; → rya
  cout << str.find("ry"); → 1

  ↳ gives index of the first iteration.
  Incase the find element couldn't find any similar string it would return [string::npos]

- acd < bcd
  ↳ program would check the lexicographical order by order for each character.

- getline (cin, str)
  ↳ unless we go to next line, it holds all char.

  getline (cin, str, '$')
  ↳ It would stop after placing a $ symbol.

May 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

- for(int i=0; i<str.length(); i++)
  cout << str[i];

for(char x : str)
  cout << x;

for(auto it = str.begin(); it != str.end(); it++)
  cout << (*it);

↳ ways to traverse the string.

26 Tuesday

*Q: Index of pattern

txt = "abcd ipcda"
pat = "cd"

O/P: 2 7

Sol":
```
void patsearch(string txt, string pat)
{
  int pos = txt.find(pat);
  while (pos != string::npos)
  {
    cout << pos << " ";
    pos = txt.find(pat, pos+1);
  }
}
```

July 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

June 2018

Wednesday 27

* Problem: Digits after decimal point.

no = 12.385

O/P → 385

Soln:

```
string digitsafterdecimal(string no)
{
    int pos = no.find('.');
    if (pos == str::npos)
        return "";
    else
        return no.substr(pos+1);
}
```

Thursday 28

* Problem: find one extra character

s1 = abcd

s2 = abecd

O/P → e

Soln:

```
char findextra(string s1, string s2)
{   sort(s1.begin(), s1.end());
    sort(s2.begin(), s2.end());
    int n = s1.length();
    for(int i=0; i<n; i++)
        if(s1[i] != s2[i])
            return s2[i];
    return s2[n]
}
```

METHOD 1
n log n

29 2018 June
Friday

May 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

METHOD-2 (use counting)

```
char findextra(string s1, string s2)
{
    int count[256] = {0}
    for(char x : s2)
        count[x]++;
    for(char x : s1)
        count[x]--;
    for(int i=0; i=256; i++)
        if(count[i] == -1)
            return (char)i;
    return 0; }
```

30 Saturday

METHOD-3 (using Bitwise)

```
char findextra(string s1, string s2)
{   int n = s1.length();

    for(int i=0; i<n; i++)
        res = res ^ s1[i] ^ s2[i];

    res = res ^ s2[n] → for the extra
    return (char)res;   character in
}                       s2
```

August 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

July 2018

Sunday 1

*Problem : To check it's a pangram

str = "The Quick Brown fox jumps over the lazy dog"

Soln:

```
bool isPangram(string s)
{
    int n = s.length();
    if (n < 26)
        return false;

    bool visited[26] = {0};

    for(int i=0; i<n; i++)
    {   char x = s[i];
```

Monday 2

```
        if (x >= 'a' && x <= 'z')
            visited[x - 'a'] = true;

        if (x >= 'A' && x <= 'Z')
            visited[x - 'A'] = true;

        for(i=0; i<26; i++)
            if (visited[i] == false)
                return false;

        return true;
    }
```

3 2018 July
Tuesday

June 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

- `__builtin_popcount()` →
- It a GCC specific function to count set bits.

```
cout << __builtin_popcount(5);
↳ 2
```

- It takes unsigned int as an argument. i.e it works for -ve numbers too.

4 Wednesday

∴ for n = -1, it would have the largest set bit value as 32.

- `__builtin_popcountl()`
  ↳ for unsigned long int
- `__builtin_popcountll()`
  ↳ for unsigned long long int