01/Mar/2023

Char Arrays & strings

- create
- memory
- '\0'
- drawback
- length
- reverse
- replace
- palindrome
- strcmp
- strlen
- strcat
- lowercase/uppercase

- what
- diff
- find(), erase(), replace(), substr(), empty(), length(), push_back(), pop_back(),

Questions–

- Valid Palindrome
- Remove all adjacent duplicate
- Remove all occurrence of substring.
- Min. time difference
- Count palindromic substring.
  - Brute force → $O(n^3)$
  - 2-pointers
    - odd substr → $O(n^2)$
    - even substr

imp.

⊙ Sort function and custom comparator →

```
string s = "babbar"
sort(s.begin(), s.end());
cout << s << endl;
```

//o/p → aabbbr

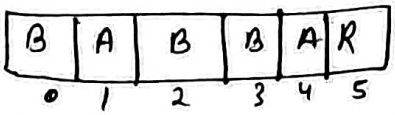
lexicographical order.

This string is converted into ascending order, but we want it in descending order.
How to do this?

- by custom comparator.

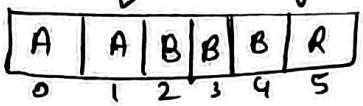
```
bool cmp(char first, char second){
    return first > second;
}
int main(){
    string s = "babbar";
    sort(s.begin(), s.end(), cmp);
    cout << s;
}
```

//o/p → rbbbaa



→ sort (s.begin(), s.end())

↓ ascending order



→ custom comparator

```
bool cmp (char x, char y){
    return x < y;   // asc order
    return x > y;   // dsc order
}
```

↳ we don't need to ~~pass~~ send these chars.

Sorting → If we have two elements then sorting is to decide which element to keep before by comparing two elements.

```
vector <int> v {5, 3, 1, 2, 4};
sort (v.begin(), v.end());
for (auto i : v)
    cout << i << " ";
} cout << endl;
```

//o/p → 1 2 3 4 5

sorted in ascending order.

```
bool compare (int a, int b){
    return a < b;   ~~return a > b;~~
}
```

→ ascending order (By default)

```
bool compare2 (int a, int b){
    return a > b;
}
```

→ descending order.

```
int main(){
    vector< int > v {5, 3, 1, 2, 4};
    sort (v.begin(), v.end(), compare);    //1 2 3 4 5
    sort (v.begin(), v.end(), compare2);   // 5 4 3 2 1
```

HW

Ques - Help Ramu coding question

Suggestion →

Whenever you have doubt about working of a function, just try ~~cout~~ printing each step of that func.

eg →
```
bool compare (int a, int b){
    cout << "first no. is:" << a << endl;
    cout << "second no. is:" << b << endl;
    return a > b;
}
main(){
    sort (v.begin(), v.end(), compare);
```

Usage →

2 character sum → we have to sort a string according to their character sum.

"love"	"babbar"	"rahul"	"sunday"	"good"

love = 12 + 15 + 22 + 5 = 54

babbar = 2 + 1 + 2 + 2 + 1 + 18 = 26

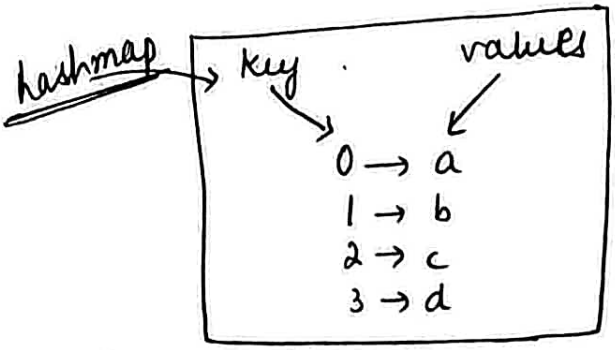
rahul = 18 + 1 + 8 + 21 + 1 = 49

sunday = 19 + 21 + 14 + 4 + 1 + 25 = 79

good = 7 + 15 + 15 + 4 = 26 + 15 = 41

ans → | "babbar" | "good" | "rahul" | "love" | "sunday" |

② Hashmap → A type of data structure in which elements are stored in form of key-value pairs.



here key is numeric. and value is character type.

```
// creation of maps →
map<int, char> meraMap;
meraMap[0] = 'a';
meraMap[1] = 'b';
meraMap[25] = 'z';
cout << "value : " << meraMap[0] << endl;   //o/p → value : a
cout << "value : " << meraMap[21] << endl;
                                  //o/p → value : ← empty.
```

Why we use map?

Because in unordered map insertion and accessing time complexity is O(1).
But here the T.C is O(log n).

auto keyword →

```
for ( auto val : ars ){
    cout << val;
}
```

By using auto we do not need to tell explicitely the datatype of element.

① maximum find

```
int maxi = INT_MIN;
```

minimum find

```
int mini = INT_MAX;
```

→ Recommend to Initialize with these.

INT_MIN and INT_MAX are macros.