ICPC
MNU COMMUNINTY

Strings

PREPERED BY: AHMED SAFFA

Agenda

- WHAT ARE STRINGS
- BASICS OPERATIONS
- BUILT-IN FUNCTIONS
- TRAVERSING & MODIFYING
- WHERE STRINGS APPEARS ?
- COMMON PITFALLS (PROBLEMS)
- PRACTICE

STRINGS

STRINGS

What is a String?

A collection of characters (letters, digits, symbols, spaces)

- Stored together in memory
- Terminated (in low-level languages) by a special character like '\0' (null character in C/C++)
- Used to represent text in programming

DECLARATION

Syntax: `#include<string>`

1.

```
string Variable_Name = “Text”;
```

2.

```
string Variable_Name;
varaibel_Name = “Text”
```

3.

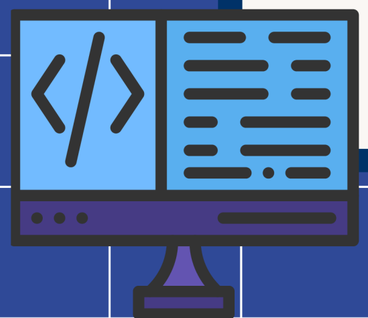
```
string Variable_Name;
cin >> Variable_Name;
```

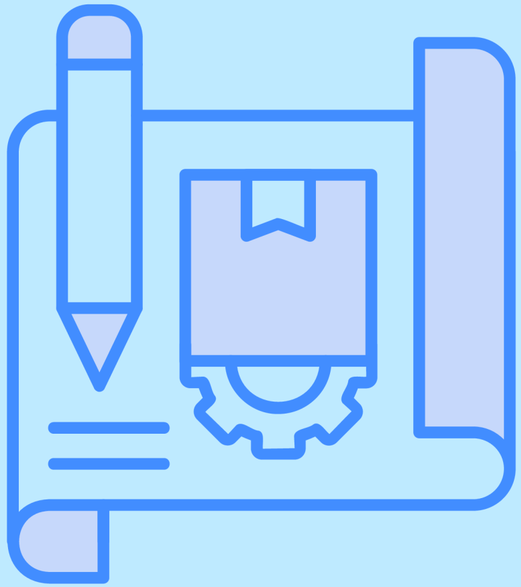
Example:

```
string word=”ICPC@MNU”;
```

```
string word;
word=”@MNU”;
```

```
string word;
cin>>word;
```





EXAMPLE

main.cpp

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

int main() {
    string name = "Ahmed";
    cout << "Hello " << name;
}

```

Output

```
Hello Ahmed
```

main.cpp

Share Run

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

int main() {
    string s1;              // empty string
    string s2 = "Ahmed";    // initialized with text

    cout << "s1: " << s1 << endl;
    cout << "s2: " << s2 << endl;
}
```

Output

```
s1:
s2: Ahmed
```

COPYING A STRING

INITIALIZATION

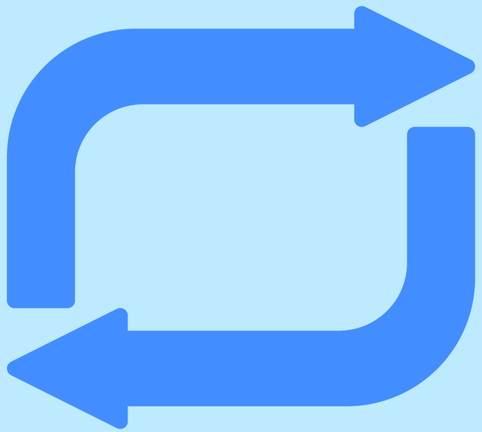
```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string name1 = "Mohamed";
    string name2 = name1; // copy

    cout << name2;
}
```

```
Output

Mohamed
```



INITIALIZING WITH REPEATED CHARACTERS

REPEATING

main.cpp

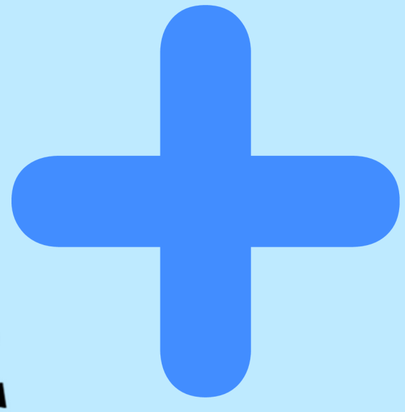
```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string stars(5, '*');
    cout << stars;
}

```

Output

```
*****
```



CONCATENATING

CONCAT

main.cpp

Share Run

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

int main() {
    string first = "Youssef";
    string last = "Ali";
    string full = first + " " + last;

    cout << full;
}

```

Output

```
Youssef Ali
```

INPUTS & OUTPUTS

CIN / COUT

main.cpp

Share Run

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

int main() {
    string name;
    cout << "Enter your first name: ";
    cin >> name;
    cout << "You entered: " << name;
}
```

Output

```
Enter your first name: Ahmed
You entered: Ahmed
```

IF YOU TYPE "AHMED SAFAA", ONLY "AHMED" WILL BE SAVED.

Output

```
Enter your first name: Ahmed Safaa
You entered: Ahmed
```

GETLINE

TO READ A FULL LINE (INCLUDING SPACES), USE GETLINE:

main.cpp

Share Run

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

int main() {
    string fullname;
    cout << "Enter your full name: ";
    getline(cin, fullname);
    cout << "Your full name is: " << fullname;
}
```

Output

```
Enter your full name: Mohamed Ali
Your full name is: Mohamed Ali
```

CIN / COUT & GETLINE

SOMETIMES GETLINE CAN ACT WEIRD IF YOU USED CIN BEFORE IT, BECAUSE CIN LEAVES A NEWLINE (\N) IN THE BUFFER.
TO FIX THIS, USE CIN.IGNORE():

main.cpp

Share Run

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

int main() {
    int age;
    string name;

    cout << "Enter your age: ";
    cin >> age;

    cin.ignore(); // clear leftover newline

    cout << "Enter your full name: ";
    getline(cin, name);

    cout << "Age: " << age << ", Name: " << name;
}
```

Output

```
Enter your age: 20
Enter your full name: Youssef Bahaa
Age: 20, Name: Youssef Bahaa
```

BUILT-IN FUNCTIONS

STR.SIZE() / STR.LENGTH()

RETURNS THE NUMBER OF CHARACTERS IN THE STRING.

main.cpp

Share Run

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

int main() {
    string name = "Ahmed";
    cout << name.size() << endl;    // 5
    cout << name.length() << endl;  // 5

}
```

Output

```
5
5
```

STR.PUSH_BACK(CHAR)

ADDS A CHARACTER AT THE END OF THE STRING.

main.cpp

Share Run

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

int main() {
    string name = "Ali";
    name.push_back('!');
    cout << name;
}
```

Output

```
Ali!
```

STR.POP_BACK()

REMOVES THE LAST CHARACTER OF THE STRING.

main.cpp

Share Run

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

int main() {
    string name = "Mohamed";
    name.pop_back();
    cout << name;
}
```

Output

```
Mohame
```

STR.POP_BACK()

REMOVES THE LAST CHARACTER OF THE STRING.

main.cpp

Share Run

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

int main() {
    string name = "Mohamed";
    name.pop_back();
    cout << name;
}
```

Output

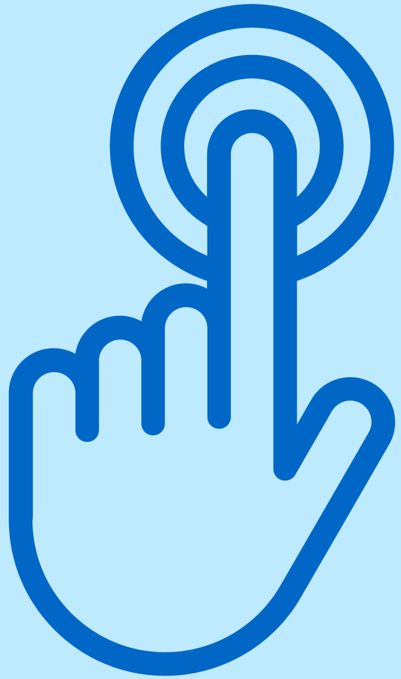
```
Mohame
```

TRAVERSING & MODIFYING

ACCESSING STRINGS

Strings are arrays of characters, so you can access each character and loop through them.

1. Accessing Characters
   - Using [ ] operator
   - Using .at() method (safer — throws error if index is invalid)
2. Changing Characters
3. Iterating Using a For Loop
   - Index-based loop
   - Range-based for loop (simpler)
4. Iterating Backwards



ACCESSING CHARACTERS

USING [ ] OPERATOR

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

int main() {
    string name = "Ahmed";
    cout << name[0];   // A
    cout << name[4];   // d
}

```

Output

```
Ad
```

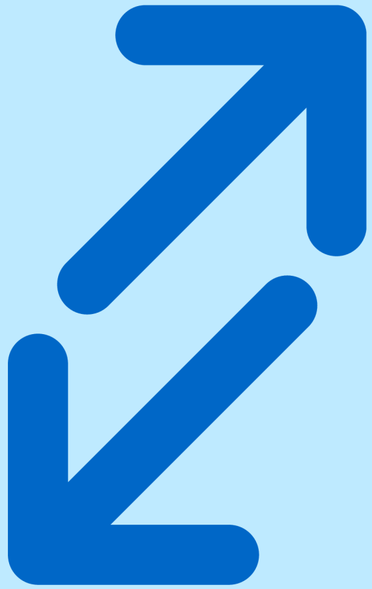
USING .AT()

main.cpp

Share Run

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

int main() {
    string name = "Ahmed";
    cout << name.at(2);  // m
}
```



CHANGING CHARACTERS

CHANGING

YOU CAN MODIFY A CHARACTER BY ACCESSING IT WITH [ ] OR .AT():

main.cpp

Share Run

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

int main() {
    string name = "Mohamed";
    name[0] = 'A';          // change 'M' to 'A'
    cout << name;           // Aohamed
}

```

Output

```
Aohamed

=== Code Execution Successful ===
```

ITERATING USING A FOR LOOP

INDEX-BASED LOOP

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

int main() {
    string name = "Ali";
    for(int i = 0; i < name.size(); i++) {
    cout << name[i] << " ";
    }
}
```

RANGE-BASED FOR LOOP

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

int main() {
    string name = "Ali";
    for(char c : name) {
    cout << c << " ";
    }
}
```

ITERATING BACKWARDS

BACKWARS

main.cpp

Share Run

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

int main() {
    string name = "Ali";
    for(int i = name.size() - 1; i >= 0; i--) {
    cout << name[i];
    }
}
```

Output

```
ilA
```

WHERE STRINGS APPEARS

- Pattern Matching
- Counting & Frequency
- Palindromes
- Searching & Parsing
- Hashing
- Prefix/Suffix Problems
- Lexicographic Order
- String Periodicity
- Sliding Window Problems
- Multi-Pattern Search
- Tries
- Dynamic Programming on Strings
- Distinct Substrings / Suffix Structures

COMMON PITFALLS

String Pitfalls

- Off-by-one errors (wrong start/end index).
- Accessing characters without checking s.size().
- Forgetting to handle empty strings ("").
- Mixing cin and getline() without clearing buffer.
- Using cin >> s for sentences (stops at space).
- Case sensitivity issues ("abc" ≠ "ABC").
- Wrong use of substr() (invalid start/length).
- Repeated s = s + c inside loops (slow).
- Forgetting to reset frequency arrays between test cases.
- Assuming input has no spaces, tabs, or special characters.
- Accessing s[s.length()] (out of bounds).
- Forgetting null terminator when using C-style char[].

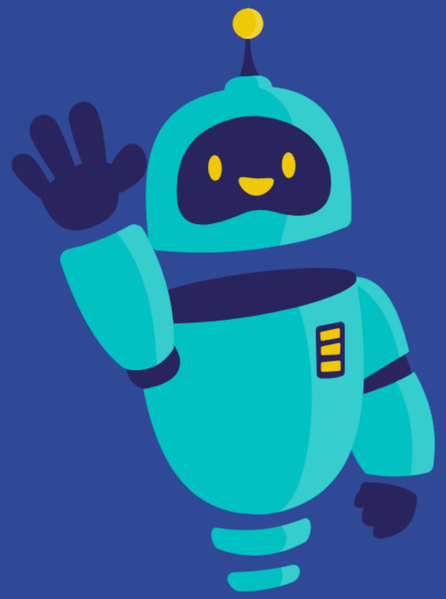
PROBLEMS

PROBLEMS

SOLUTIONS

CODEFORCE TIME !

THANK YOU!



FOLLOW US !