**String Transformation**

Consider a **string** s to be transformed -

*String Lowercase to Uppercase :*

**transform**(s.begin(), s.end(), s.begin(), ::toupper);

*String Uppercase to Lowercase :*

**transform**(s.begin(), s.end(), s.begin(), ::tolower);

**NOTE**

For Character (**char**) Transformation :

**toupper**(char) - Turn char to upper-case letter

**tolower**(char) - Turn char to lower-case letter

**Swapping two values**

Consider a string s1 = “ABC”;

*Sample Swapping E.g* :

**swap**(s1[x],s1[y]); // STL way of Swapping Two Values

**NOTE**

* The Swapping Method Described Above can be used for Integer Array as well
* Swap method comes under <utility> header file

**Converting string to integer**

To convert a string to a digit in C++ STL, you can use the stoi() function.

Here's an example:

string str = "123";

int num = **stoi**(str);

**NOTE**

* stoi() method comes under <string> header file
* If the input string is not a valid integer, stoi() will throw an invalid\_argument exception.

To handle this, you can use a try-catch block:

#include <iostream>

#include <string>

#include <stdexcept>

using namespace std;

int main() {

string str = "abc";

try {

int num = stoi(str);

cout << "String: " << str << endl;

cout << "Number: " << num << endl;

} catch (const invalid\_argument& e) {

cout << "Error: " << e.what() << endl;

}

return 0;

}

**Reverse a String**

Consider a String s1= “Hello”;

*Reversing Method ( STL Usage ) :* ***reverse****(s1.begin(), s1.end());*

*Reversing Method ( Two Pointer Approach ) :*

void reverse(string &s1, int len, int i){

if(i >= len)

return;

swap(s1[i],s1[len]);

reverse(s1, len-1;i+1);

}

**Comparing Two Strings**

Consider Two String s1 and s2, s1 = “Hello”, s2 = “World”

*Syntax to Compare s1 and s2* : **(s1 == s2)** also works the same

s1.**compare**(s2); *OR* s2.**compare**(s1);

- If the result is 0, Then strings are Matched

**Compare string for lexicographical order (dictionary order)**

std::string str1 = "Apple";

std::string str2 = "Appla";

/ / This is enough to check who appears earlier in lexicographical order

if (str1 <= str2) {

cout << "str1 comes before str2 in dictionary order";

} else {

cout << "str2 comes before or is equal to str1";

}

* The Following Technique is capable of
  + Distinguish between Uppercase and lowercase letters in lexicographical order (i.e Uppercase comes earlier than lowercase letters)

**Truncate a String using substr()**

**string**.substr(x, y): The substr() function returns a new string that is a substring of the original string.

Syntax: string substr (size\_t pos, size\_t len) const;

Parameters:

pos: Position of the first character to be copied.

len: Length of the sub-string.

size\_t: It is an unsigned integral type.

Here's an example code that truncates a string into two halves:

string str = "Hello, World!";

int len = str.length();

int midpoint = len / 2;

string first\_half = str.substr(0, midpoint);

// here midpoint is length of substring

string second\_half = str.substr(midpoint);

**Convert integer (int) to string**

**to\_string(int)** - This method can be used to convert int to string by passing int to it directly or through a variable.

**Verify Whether a Character is Alphanumeric**

**isalnum()** - Method that returns non-zero if the ASCII value passed in has a character equivalent to a letter or number else returns zero.

Prototype: int alnum(int ch);

Header File: cctype (C++), ctype.h (C)

**Find Min/Max Element Between Two Values**

**min**(value1, value2) - Returns Element whichever is minimum

**max**(value1, value2) - Returns Element whichever is maximum

**Obtain the Posssible min/max value of INT**

**INT\_MIN** - Returns minimum integer value

**INT\_MAX** - Returns maximum integer value

**istringstream()**

**istringstream** is a class in the C++ Standard Template Library (STL) that allows for easy input parsing and conversion of strings into other data types. It is included in the <**sstream**> header file.

The **istringstream** class takes a string as its input and provides methods to extract individual tokens from the string separated by blank-space. This is useful for **parsing** text files or data from a network connection.

Some of the key features of istringstream include:

1. **>> operator**: The >> operator can be used to extract individual tokens from the string. For example, **iss >> num** will extract the next token from the string and store it in the variable num.

2. **getline method**: The **getline** method can be used to extract an entire line of text from the string.

3. **Conversion functions**: The istringstream class provides several built-in conversion functions, such as **stoi** and **stod**, which can be used to convert strings to **integers** or **doubles**, respectively.

Here's an example of how to use istringstream to parse a string:

#include <sstream>

#include <string>

#include <iostream>

using namespace std;

int main() {

std::string input = "42 3.14 hello world";

std::istringstream iss(input);

// Another method to achieve same thing as above

/\*

std::istringstream iss;

iss.str(input);

\*/

int num;

double dnum;

std::string str;

std::string str2;

std::string str3;

iss >> num >> dnum >> str >> str2 >> str3;

std::cout << "num = " << num << std::endl; // 42

std::cout << "dnum = " << dnum << std::endl; // 3.14

std::cout << "str = " << str << std::endl; // hello

std::cout << "str2 = " << str2 << std::endl; // world

std::cout << "str3 = " << str3 << std::endl; //

return 0;

}

**NOTE**

* We can Initialize or reuse **istringstream** by this method
  + Step 1: ss.clear();
  + Step 2: ss.str("");

**Check whether the char (character) is a digit**

**isdigit(char)** - we can use this method to check whether the **char** we have is an equivalent digit or not.

For example (consider char and a string )

char x = ‘1’;

string str = ‘a1’;

cout << isdigit(x); // true

cout << isdigit(str[0]); // false

cout << isdigit(str[1]); // true

**NOTE**

* We can only supply **char** variable to isdigit(), and not a **string**
* Declared in <**ctype.h**> header file

**Check Datatype of a variable**

**typeid(variable).name()** : returns type of the supplied value or variable.

Sample use-case examples:

string x = “abc”;

int i = 1;

char c = ‘a’;

cout << typeid(x[0]).name() << endl; // c

cout << typeid(x).name() << endl; // … string

cout << typeid(c).name() << endl; // c

cout << typeid(i).name() << endl; // i

**NOTE**

* It can be used in debugging purposes
* Defined in <**typeinfo**> header file

**Find the absolute value / mod of a difference**

**fabs()** : fabs can be used to calculate absolute value of **floating integer types** and is defined in <**math.h**> header file.

**abs()** : abs can be used to calculate absolute value of **integer types** and is defined in <**stdlib.h**> header file

**Key Findings of abs( )**

* abs will not work properly/produce errors if you do something like this when computing absolute length of strings.

string x = “abc”, y = “defgh:;

abs(x.length() - y.length()); // Error

* Instead, take length outside of abs and then directly supply int values,

check the next page for suggestions on the same.

Example valid use-case:

| int a = 5, b = 7; | string a = “abc”, b = “defgh”; |
| --- | --- |
| abs(a - b); // 2 | int s1 = a.length(); |
| abs(b - a); // 2 | int s2 = b.length(); |
|  | abs(s1 - s2); // 2 |
|  | abs(s2 - s1); // 2 |

**String Concatenation**

There are many methods to concatenate two strings and here are some of the key methods:

1. **+ operator**

E.g string a = “ab”, b = “cd”;

cout << a + b; // abcd

1. **append( )**

E.g string a = “ab”, b=”cd”

a.append(b); // abcd

* **Operator Key Findings**
* The + Operator will not work if concatenation is done in this fashion

1. string s = ““ + ‘a’; // Error
2. string s = “abc”;

string x = ““ + s.back(); // Error

* Try Instead,

string s = “abc”;

string x = “”;

x = x + s.back();

* In above statements we are trying to convert a **char** type variable to **string** type variable (‘a’ and s.back() are both **char** type variable)

**Converting a single digit character (char) to its integer (int) equivalent**

Suppose we have a **single digit** represented as a **character** **(0-9)**. When we try to print the character **'0'**, it is actually represented as the **ASCII** value **48**. We want to obtain the integer equivalent of the digit, i.e. 0 for '0', 1 for '1', 2 for '2', and so on up to 9.

To achieve this, we can use a simple trick:

**char** ch = '5'; // any number

**int** i = **ch** - '0'; // **implicit type casting** occurs here

**NOTE**

* Subtracting the ASCII value of **'0'** from the character gives us the corresponding integer value of the digit. Here, an implicit type-casting occurs, which converts the character to its corresponding int value.
* It is important to note that **explicit type-casting**, such as **(int)ch**, will not work to convert a single digit represented as a character to its integer value. This is because type-casting converts the ASCII value of the character to an integer, which will not yield the desired result.

**Essential ASCII values**

* **‘\0’** : 0 (A **null** terminator)
* **‘0’ - ‘9’ :** 48 - 57 { 48 = ‘0’, 49 = ‘1’, … 57 = ‘9’ }
* **‘ ‘** (blank-space) **:** 32
* **a - z :** 97 - 122 { 97 = a, 98 = b, … 122 = z }
* **A - Z** : 65 - 90 { 65 = A, 66 = B, … 90 = Z }
* � : 128 - 255 { 128 = � , 129 = � , … 255 = � }

**NOTE**

* ASCII values are in the range **0 - 255**. After 255, values starts repeating i.e { 256 == 0, 257 == 1, … A == 321, … a == 353, … }
* **Printable characters** are represented by ASCII values in the range of **0-127**, while **Non-Printable characters** are represented by ASCII values in the range of **128-255**.

**Vector to String**

vector<int> cont = {1, 2, 3, 4, 5};

string str = string(cont.begin(), cont.end());

for(auto x : str)

cout << x; // 1 2 3 4 5

**NOTE**

* **string(cont.begin(), cont.end())** is a constructor call to create a new **string** object from a **pair of iterators** cont.begin() and cont.end()

**Length of a string and Arithmetic Operations**

**sizeof**(str)/**sizeof**(char) : It is classic way to get the length of a string/array in C/C++, defined in the form of a **character array** char str [ ] = “i am a string \0”

string.**length() : length()** method is used to retrieve the current length of the **string** and is defined in <**string**> header  **(**C++ Only)

**NOTE**

* It is **not advisable** to perform **arithmetic operations** directly when computing the length of a string. Instead, it is recommended to store the length in a variable and then use that variable for any further arithmetic operations.
* For example, rather than computing (string.length() - k), save the length in a variable and then perform the necessary arithmetic using the variable.

**Convert Iterator to a index value**

There is one of the situations when we have an Iterator and for some reason we want its equivalent index value.

For example, consider a string email = “[username@domain.com](mailto:username@domain.com)”

* Our goal is to extract **local name** and **domain name** out of this string.
* Using the substr**()** method we do it easily and **find()** method can be used to get an iterator to **@** which separates local name and domain name.
* auto iterator = find(email.begin(), email.end(), ‘@’)

**iterator** now points to @ in the **email** string but here’s the problem that arises when we want to supply it to **substr**() as it takes **int** value followed by **length** instead of **string iterator.**

* To convert Iterator to index we can simply do **(it - email.begin())**
* **NOTE:** email.begin() is equivalent to 0 index and that's hypothetically being subtracted from iterator **it.**
* **(it - email.end()) is wrong way -** don't get confused