Topic4(User Input and Constants)

C++ User Input

> Now we will use `cin` to get user input. `cin` is a predefined variable that reads data from the keyboard with the extraction operator(`>>`).

> In the following example, the user can input a number, which is stored in the variable `x` . Then we print the value of `x`:

```
int x;
cout << "Type a number : ";        // Type a number and press enter
cin >> x;                          // Get user input from the keyboard
cout << "Your number is : " << x;  // Display the input value
```

> NOTE : `cout` is pronounced “see-out”. Used for **output**, and uses the insertion operator (`<<`)

> **NOTE :** `cin` is pronounced “see-in”. Used for **input**, and uses the extraction operator (`>>`)

Why would we call cin.clear() and cin.ignore() after reading input?

> The `cin.clear()` clears the error flag on `cin` (so that future I/O operations will works correctly).

> The `cin.ignore(1000, ‘\n’)` help you to take input after taking one input print it earlier.

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

int main(int argc, char const *argv[])
{
    int n, x;
    cin >> n;
    cout << n << endl;

    cin.clear();
    cin.ignore(1000, '\n');

    cin >> x;
    cout << cin.fail() <<endl;
    cout << x;
    return 0;
}
```

Creating a Simple Calculator

> In this example, the user must input two numbers. Then we print the sum by calculating (adding) the two numbers :

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

int main(int argc, char const *argv[])
{
    int x, y, sum;
    cout << " Type a number : ";
    cin >> x;
    cout << " Type another number : ";
    cin >> y;
    sum = x + y;
    cout << " Sum is : " << sum;
    return 0;
}
```

C++ Constants

> When you do not want others (or yourself) to change existing variable values, use the `const` keyword (this will declare the variable as “constant”, which means **unchangeable and read-only**):

```
const int myNum = 15;
myNum = 10;
```

> You should always declare the variable as constant when you have values that are unlikely to change:

```
const int minutesPerHour = 60;
const float PI = 3.14;
```

C++ Strings

> Strings are used for storing text. A `string` variable contains a collection of characters surrounded by double quotes :

> **Example**
> Create a variable of type `string` and assign it a value :
>
> ```
> string greeting = "Hello";
> ```

> To use strings, you must include an additional header file in the source code, the `<string>` library :

> **Example**
>
> ```
> // Include the string library
> #include <string>
>
> // Create a string variable
> string greeting = "Hello";
> ```

String Concatenation

> The `+` operator can be used between strings to add them together to make a new string. This is called **concatenation** :

```
string firstName = "John";
string lastName = "Doe";
string fullName = firstName + lastName;
cout << fullName;
```

> In the example above, we added a space after firstName to create a space between John and Doe on output. However, you could also add a space with quotes (`""` or `''`) :

```
string firstName = "John";
string lastName = "Doe";
string fullName = firstName + " " + lastName;
cout << fullName;
```

Append

> A string in C++ is actually an object, which contain functions that can perform certain operations on string. For example, you can also concatenate strings with the `append()` function :

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName.append(lastName);
cout << fullName;
```

Adding Numbers and Strings

> WARNING !
> C++ uses the + operator for both addition and concatenation.
> Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

```
int x = 10;
int y = 20;
int z = x + y;      // z will be 30 (an integer)
```

If you add two strings, the result will be a string concatenation:

```
string x = "10";
string y = "20";
string z = x + y;   // z will be 1020 (a string)
```

If you try to add a number to a string, an error occurs :

```
string x = "10";
int y = 20;
string z = x + y;
```

String Length

To get the length of a string, use the `length()` function:

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "The length of the txt string is : " << txt.length();
```

> NOTE : You might see some C++ programs that use the `size()` function to get the length of a string. This is just an alias of `length()`. It is completely up to you if you if you want to use `length()` or `size()`:

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "The length of the txt string is : " << txt.size();
```

Access Strings

> You can access the characters in a string by referring to its index number inside square brackets `[]`.

This example prints the first character in myString:

```
string myString = "Hello";
cout << myString[0];
```

> NOTE : String indexes start with 0: [0] is the first character. [1] is the second character, etc.

This example prints the second character in myString:

```
string myString = "Hello";
cout << myString[1];
```

Change String Characters

To change the value of a specific character in a string, refer to the index number, and use single quotes:

```
string myString = "Hello";
myString[0] = 'J';
cout << myString;
```

Strings - Special Characters

> Because strings must be written within quotes, C++ will misunderstand this string, and generate an error:
>
> ```
> string txt = "We are the so-called "Vikings" from the north.";
> ```

The backslash(`\`) escape character turns special characters into string characters:

Escape character	Result	Description
\’	‘	Single quote
\”	“	Double quote
\\	\	Backslash

The sequence `\"` inserts a double quote in a string:

```
string txt = "We are the so-called \"Vikings\" from the north.";
```

The sequence `\'` inserts a single quote in a string:

```
string txt = "It\'s alright.";
```

The sequence `\\` inserts a single backslash in a string:

```
string txt = "The character \\ is called backslash.";
```

Other popular escape characters in C++ are:

Escape Character	Result	Example
`\n`	New Line	`string txt = "Hello\nWorld!";`
`\t`	Tab	`string txt = "Hello\tWorld!";`

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

int main() {
  string txt = "Hello\nWorld!";
  cout << txt;
  return 0;
}
```

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

int main() {
  string txt = "Hello\tWorld!";
  cout << txt;
  return 0;
}
```

C++ User Input Strings

It is possible to use the extraction operator `>>` on `cin` to display a string entered by a user:

```
string firstName;
cout << "Type your first name: ";
cin >> firstName; // get user input from the keyboard
cout << "Your name is: " << firstName;
```

> However, `cin` considers a space (whitespace, tabs, etc) as a terminating character, which means that it can only display a single word (even if you type many words):
>
> ```
> string fullName;
> cout << "Type your full name: ";
> cin >> fullName;
> cout << "Your name is: " << fullName;
> ```
>
> From the example above, you would expect the program to print “John Doe”, but it only prints “John”.

That’s why, when working with strings, we often use the `getline()` function to read a line of text. It takes `cin` as the first parameter, and the string variable as second:

```
string fullName;
cout << "Type your full Name : ";
getline(cin, fullName);
cout << "Your name is : " << fullName;
```

C++ String Namespace

Omitting Namespace

You might see some C++ programs that runs without the standard namespace library. The `using namespace std` line can be omitted and replaced with the `std` keyword, followed by the `::` operator for `string` (and `cout`) objects:

```
#include <iostream>
#include <string>

int main() {
  std::string greeting = "Hello";
  std::cout << greeting;
  return 0;
}
```

> NOTE : It is up to you if you want to include the standard namespace library or not.

C++ Math

> C++ has many function that allows you to perform mathematical tasks on numbers.

Max and Min

The max(x, y) function can be used to find the highest value of x and y:

```
cout << max(5, 10);
```

And the min(x, y) function can be used to find the lowest value of x and y:

```
cout << min(5, 10);
```

C++ <cmath> Header

> Other functions, such as `sqrt` (square root), `round` (rounds a number) and `log` (natural logarithm), can be found in the `<cmath>` header file:

```
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main(int argc, char const *argv[])
{
    /* code */

    cout << sqrt(64) << endl;
    cout << round(2.6) << endl;
    cout << log(2) << endl;

    return 0;
}
```

Other Math Functions

A list of other popular Math functions (from the `<cmath>` library) can be found in the table below:

Function	Description
`abs(x)`	Returns the absolute value of x
`acos(x)`	Returns the arccosine of x
`asin(x)`	Returns the arcsine of x
`atan(x)`	Returns the arctangent of x
`cbrt(x)`	Returns the cube root of x
`ceil(x)`	Returns the value of x rounded up to its nearest integer
`cos(x)`	Returns the cosine of x
`cosh(x)`	Returns the hyperbolic cosine of x
`exp(x)`	Returns the value of E^x
`expm1(x)`	Returns e^x - 1
`fabs(x)`	Returns the absolute value of a floating x
`fdim(x, y)`	Returns the positive difference between x and y
`floor(x)`	Returns the value of x rounded down to its nearest integer
`hypot(x, y)`	Returns sqrt(X^2 + Y^2) without intermediate overflow or underflow
`fma(x, y, z)`	Returns X*Y+Z without losing precision
`fmax(x, y)`	Returns the highest value of a floating x and y
`fmin(x, y)`	Returns the lowest value of a floating x and y
`fmod(x, y)`	Returns the floating point remainder of x / y
`pow(x, y)`	Returns the value of x to the power of y
`sin(x)`	Returns the sine of x (x is in radians)
`sinh(x)`	Returns the hyperbolic sine of a double value
`tan(x)`	Returns the tangent of an angle
`tanh(x)`	Returns the hyperbolic tangent of a double value