C++

CPP-01

(Variables, Basic Types, Control Structures)

⇒ Compiling hello_world.cpp file

```
c++ -std=c++11 -o hello_world hello_world.cpp
```

↓

(name of our program in the end)

★ Declaring Variables

⇒ Variable declaration always follows pattern:

```
<TYPE> <NAME> [= <VALUE>];
```

→ Every variable has a type
→ Variables cannot change their type
→ Always initialize variables if you can.

★ Naming variables

→ Name must start with a letter.
→ Give variables meaningful names.
→ Don't be afraid to use a long name.
→ Don't include type in the name.
→ Don't use negation in the name.
→ GOOGLE_STYLE name variables in (snake_case) all lowercase, underscores separated words.
→ C++ is case sensitive

★ Built-in types

① bool → Either true or false
② char → Single character
③ int → Integer number
④ short → Short number
⑤ long → Long number
⑥ float → Single precision float (0.01f)
⑦ double → Double precision float
⑧
```
auto someint = 13;          [int]
auto some_float = 13.0f;    [float]
auto some_double = 13.0;    [double]
```

★ Operations on arithmetic types

⇒ All character, integer and floating point types are arithmetic.

⇒ Arithmetic operations: +, −, *, /

⇒ Comparisions <, >, <=, >=, ==
returns bool.

⇒ Avoid == for floating point types

⇒ a += 1 ⇔ a = a + 1, Same for −=, *=, /= etc.

★ Some additional Operations

⇒ Boolean variables have logical operations
or: ||, and: &&, Not: !

→ / is integer division: 7/3 == 2
→ % is modulo division: 7 % 3 == 1
→ a++ ⇔ ++a ⇔ a+=1
→ a-- ⇔ --a ⇔ a-=1

(first updated then return a value) — faster recommended

(first return a value then updated a)

★ Strings (Part of std library)

⇒ #include <string> to use std::string.
⇒ Concatenate strings with +
⇒ Check if str is empty with str.empty()
⇒ Works out of the box with I/O stream

★ Use std::array for fixed size collection of items

⇒ #include <array> to use std::array
⇒ Stores collection of items of same type
⇒ Create from data:

```
std::array<float, 3> arr = {1.0f, 2.0f, 3.0f};
```

⇒ Access items with arr[i] indexing starts with 0
⇒ Number of stored items: arr.size()
⇒ Useful access aliases:
- First item arr.front() == arr[0]
- Last item arr.back() == arr[arr.size()-1]

★ Use std::Vector when number of items is Unknown beforewise

⇒ #include <vector> to use std::vector

⇒ Vectors are implemented as a dynamic table.

⇒ Access stored item just like in std::array

⇒ Add a new item in one of two ways:

- vec.emplace_back(value)
  [Preferred, C++11]

- vec.push_back(value)
  [Historically better known]

"Consider it to be a default container to store collection of items of any same type"

★ Optimize vector resizing

⇒ Many push_back/emplace_back operations force vector to change its size many times.

⇒ reserve(n) ensures that the vector has enough memory to store n items.

⇒ The parameter n can even be approximate

⇒ This is very important optimization

```
std::vector<std::string> vec;
const int KItemNum = 100;
vec.reserve(KItemNum);
```

★ Variables live is scopes

⇒ There is a Single global scope
⇒ Local scope starts with { and ends with }.
⇒ All Variables belong to the scope where they have been declared.
⇒ All Variable die in the end of their scope.
⇒ This is the core of C++ memory system.

★ Any Variable can be Const

⇒ Use const to declare a constat
⇒ The compiler will guard it from any changes
⇒ Keyword const can be used with any type.
⇒ GOOGLE-STYLE name constants in CamelCase. Starting with a small letter K:

Example:

```
const int kSomeInt = 20;
```

⇒ const is part of type:

Variable KSomeInt has type const int

⇒ Tip: declare everything Const unless it must be changed.

★ Reference to variable

⇒ We can create a reference to any variable.

⇒ Use & to state that a variable is a reference.

example:
```
std::string& hello_ref = hello;
```

⇒ Reference is part of type:

Variable hello_ref has type string&

⇒ Whatever happens to a reference happens to the variable and vice versa.

⇒ Yields performance gain as reference avoid copying data.

★ Const with references

Example: `Const float& Ref = Original;`

★ If Statment

⇒ Used to conditionally execute code

```
if (STATEMENT) {

} else if (OTHER_STATEMENT){

}else{

}
```

⇒ STATEMENT can be any boolean expression.

* Switch statement

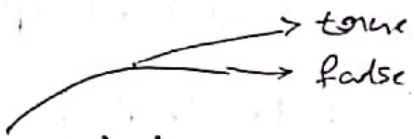
⇒ Used to Conditionally execute code.

```
switch (STATEMENT) {
    case CONST_1:
        break;
    case CONST_2:
        break;
    default:
}
```

⇒ break exits the switch block

⇒ STATEMENT usually returns int or enum value.

* While loop




```
while (STATEMENT) {
}
```

⇒ Usually used when the exact number of iterations is unknown before-wise.

* For loop

```
for (INITIAL_CONDITION; END_CONDITION; INCREMENT) {
}
```

★ Range for loop

⇒ Iterating over a standard containers like array or vector has simpler syntax.
- ⇒ Show intent with the syntax
- ⇒ Has been added in C++11

```
for (const auto& value : container) {

}
```

★ Exit loops and iterations

⇒ Use break to exit the loop.

⇒ Use continue to skip to next iteration