```cpp
#include <iostream>

int main() {
  std::cout << "Hello world" << std::endl;
  return 0;
}
```

```cpp
#include <iostream>

int main() {
  int a, b;
  std::cin >> a >> b;
  std::cout << a + b << std::endl;
  return 0;
}
```

- `std::cin`, `std::cout`, names from the standard library
- To avoid name collision: `namespace std`.

```cpp
#include <iostream>

using std::cin;

int main() {
  int a, b;
  cin >> a >> b; // std::cin -> cin
  std::cout << a + b << std::endl;
  return 0;
}
```

- `using namespace std;` introduces every name in `std`.

- integers, floating numbers, characters, strings

example

`std::string`

`#include <string>`

`std::string`

a string class abstraction

Create a string

```
string str = "Hello world";
// string str("Hello world"); equivalent
cout << str << endl;
string s1(7, 'a');
cout << s1 << endl;
string s2 = s1; // a copy
cout << s2 << endl;
```

Length of a string

```
string str = "Hello world";
cout << str.size() << std::endl;
```

Not `strlen`, not `sizeof`.

Not null-terminated.

Check whether a string is empty:

```
if (str.empty()) {
  // ...
}
```

String concatenation (`+`, `+=`)

examples

`+=` !!!

String comparison: relational and equality operators (lexicographical order)

String assignment: use `=` directly

semantic: copy (We don't need other functions like `strcpy`)

different from C-style strings:

```c
const char *s1 = /* ... */;
const char *s2 = NULL;
s2 = s1; // points to the same string
```

IO of `std::string`

```
string str;
cin >> str; // Discard leading whitespaces, read until whitespace
string line;
getline(cin, line); // Read from the current position to the first newline ('\n')
```

Conversion between `std::string` and integers:

`std::to_string`

```cpp
int ival = 42;
double dval = 3.14;
std::string s = std::to_string(ival) + std::to_string(dval);
// s == 423.14
```

`std::stoi, std::stol` ...

https://en.cppreference.com/w/cpp/string/basic_string#Numeric_conversions

We don't use `strtol` or `atoi`: They are for C-style strings.

Access by subscript:

Output uppercase letters:

```
for (std::size_t i = 0; i != s.size(); ++i)
  if (std::isupper(s[i]))
    std::cout << s[i];
std::cout << std::endl;
```

header: `<ctype.h>` -> `<cctype>`

For header files inherited from the C library, we recommend using `<cname>` instead of `name.h`. (reason?)

Names in `<cname>` are also in `namespace std`. Both `std::tolower` and `tolower` exist.

`<stdio.h>` -> `<cstdio>`, `printf` -> `std::printf`

Range-based for-loops

Output uppercase letters:

```cpp
for (std::size_t i = 0; i != s.size(); ++i)
  if (std::isupper(s[i]))
    std::cout << s[i];
std::cout << std::endl;
```

Better (modern) way:

```cpp
for (char c : s)
  if (std::isupper(c))
    std::cout << c;
std::cout << std::endl;
```