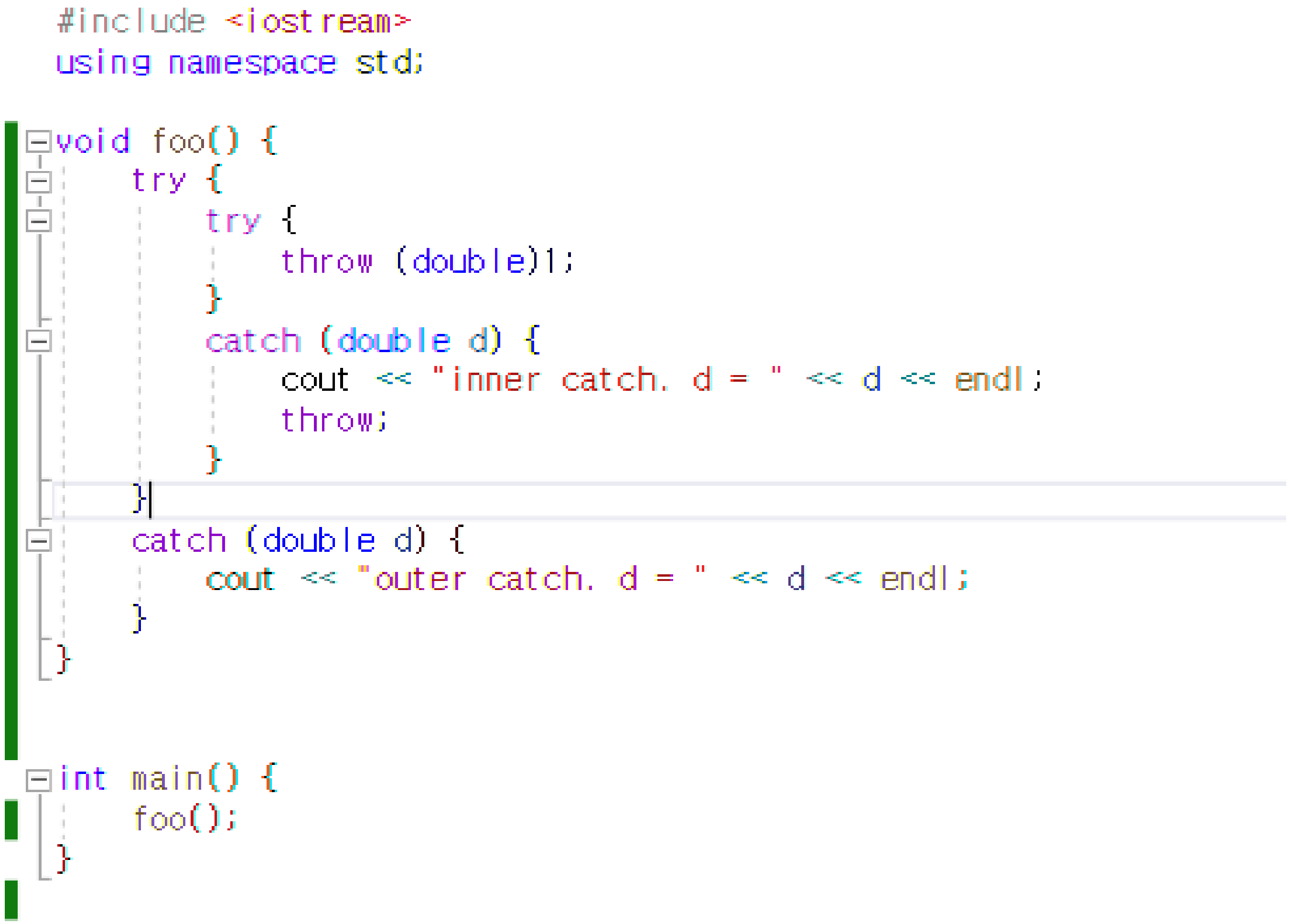
C++ Programming(Exercise) 10

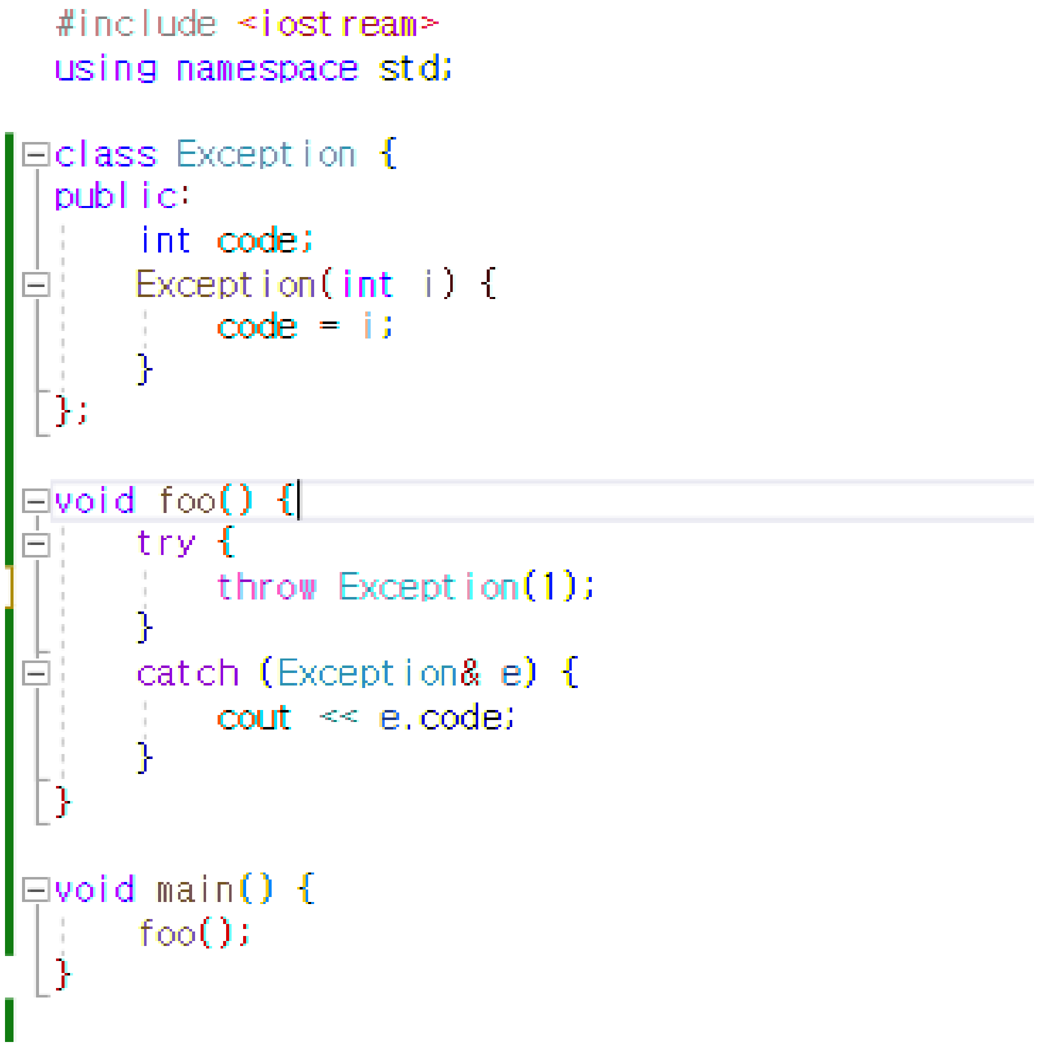
- Arne Kutzner
- Hanyang University / Seoul Korea

Nested Try-Catch



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

void foo() {
    try {
        try {
            throw (double)1;
        }
        catch (double d) {
            cout << "inner catch. d = " << d << endl;
            throw;
        }
    }
    catch (double d) {
        cout << "outer catch. d = " << d << endl;
    }
}


int main() {
    foo();
}
```

Exception Class - catch by reference



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

class Exception {
public:
    int code;
    Exception(int i) {
        code = i;
    }
};

void foo() {
    try {
        throw Exception(1);
    }
    catch (Exception& e) {
        cout << e.code;
    }
}

void main() {
    foo();
}
```

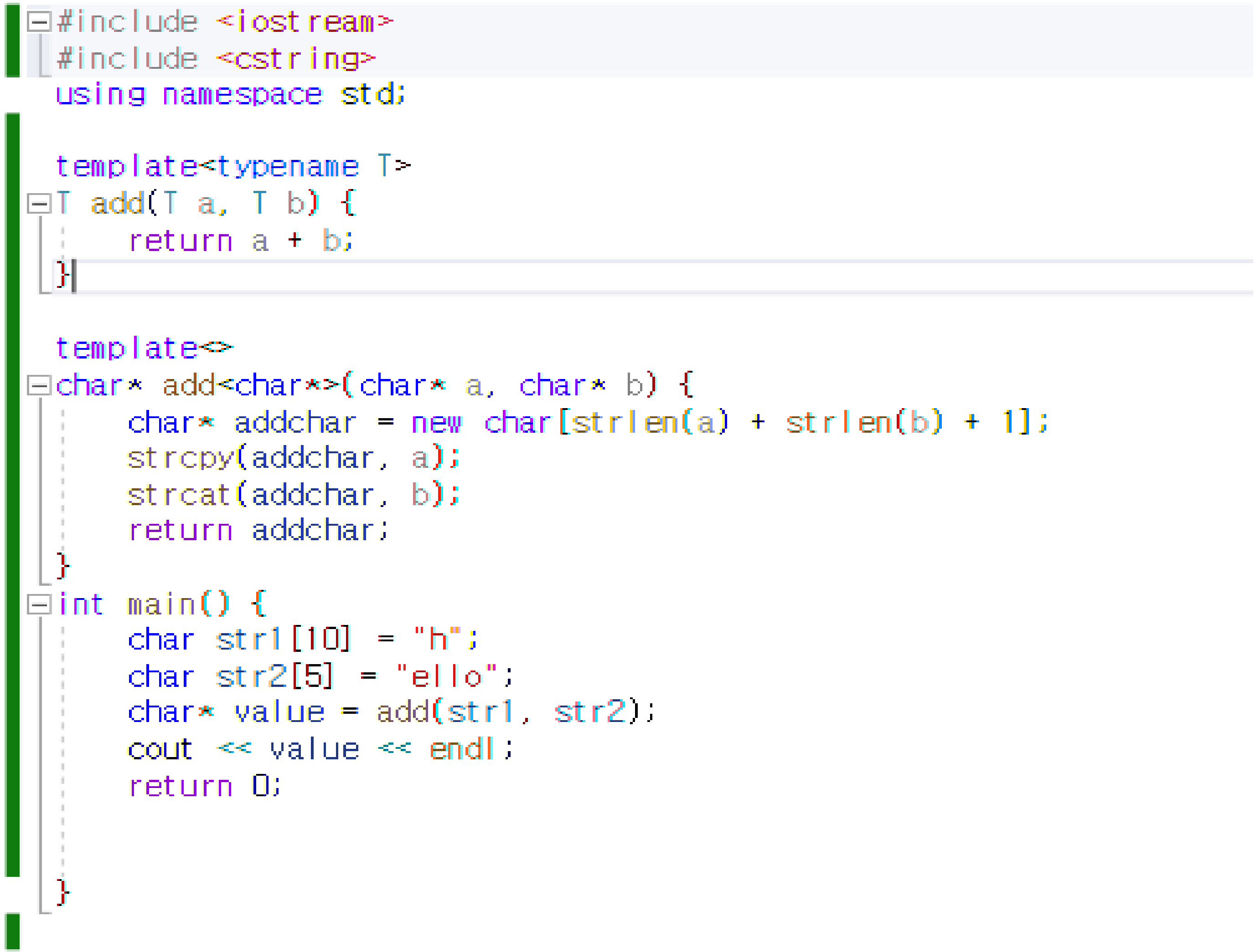
Template

- General form of Template declaration:

```
template <class identifier>
    class_definition;
or
template <typename identifier>
    class_definition;
```

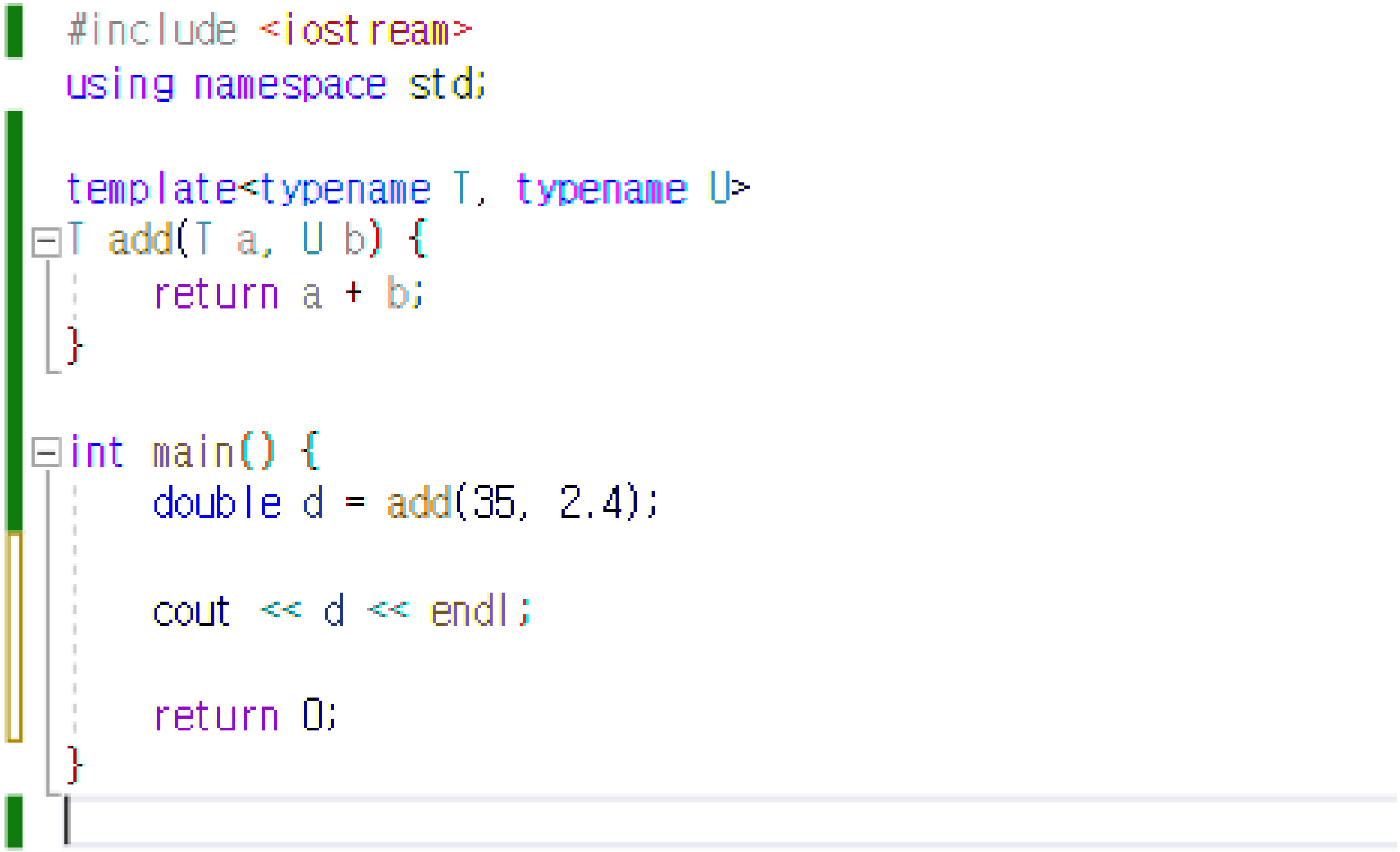
- Both forms are identical.
- With template, you don’t need to declare a function with different data types for multiple times.

Templates Specialization



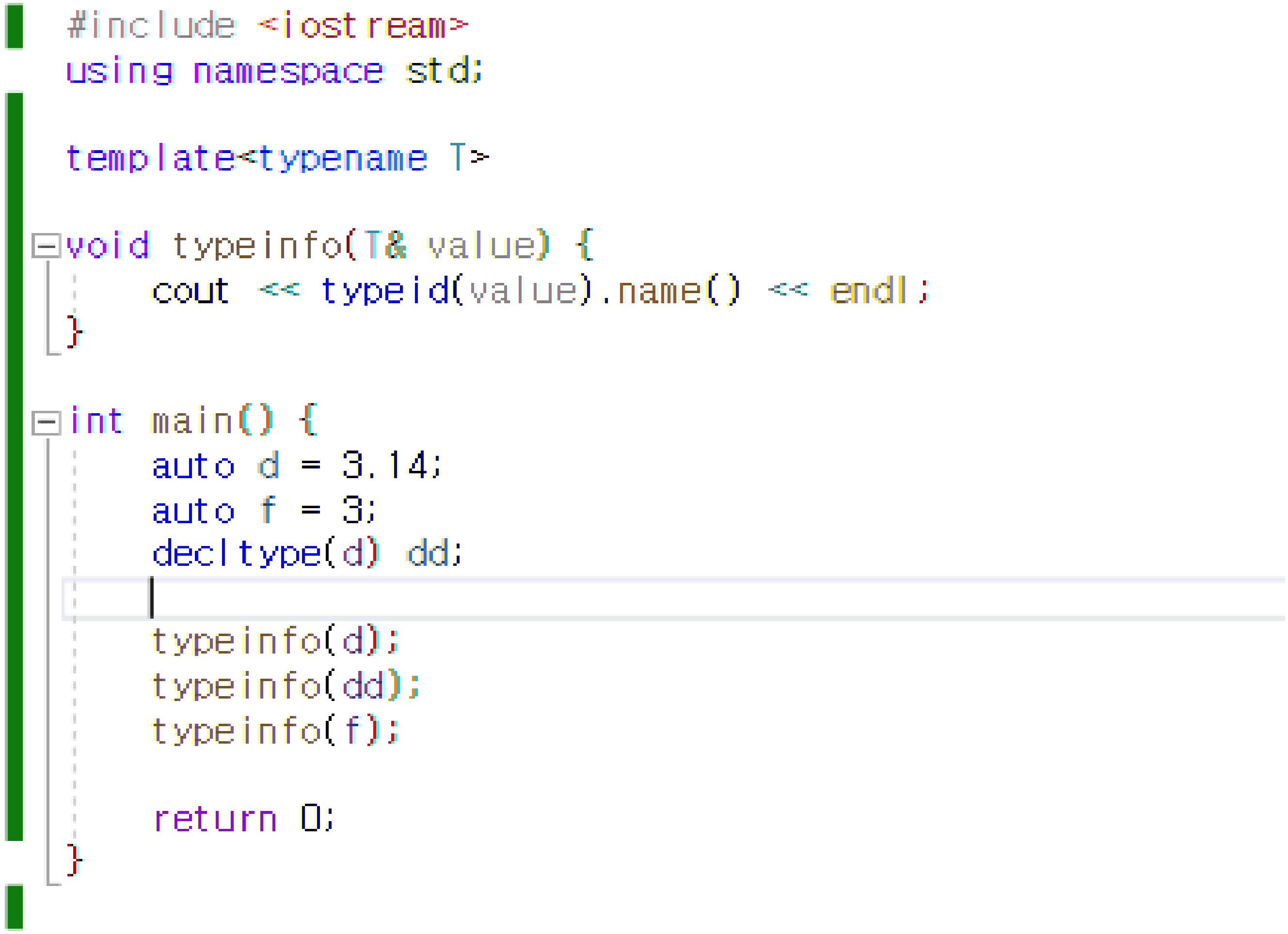
```cpp
#include <iostream>
#include <cstring>
using namespace std;

template<typename T>
T add(T a, T b) {
    return a + b;
}

template<>
char* add<char*>(char* a, char* b) {
    char* addchar = new char[strlen(a) + strlen(b) + 1];
    strcpy(addchar, a);
    strcat(addchar, b);
    return addchar;
}
int main() {
    char str1[10] = "h";
    char str2[5]  = "ello";
    char* value = add(str1, str2);
    cout << value << endl;
    return 0;


}
```

Type inference



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

template<typename T, typename U>
T add(T a, U b) {
    return a + b;
}

int main() {
    double d = add(35, 2.4);

    cout << d << endl;

    return 0;
}
```

Template, Auto &Decltype

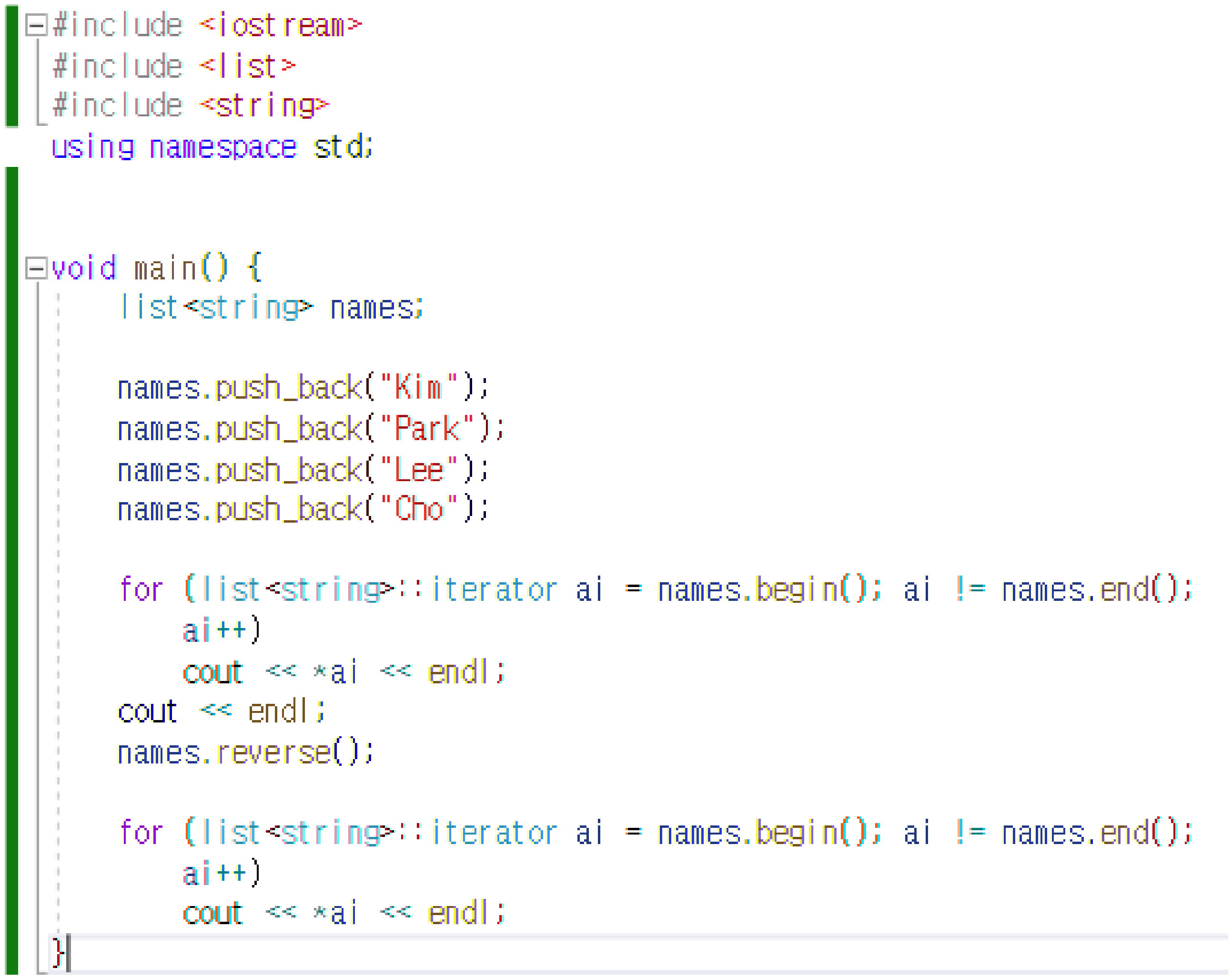


```
#include <iostream>
using namespace std;

template<typename T>

void typeinfo(T& value) {
    cout << typeid(value).name() << endl;
}

int main() {
    auto d = 3.14;
    auto f = 3;
    decltype(d) dd;

    typeinfo(d);
    typeinfo(dd);
    typeinfo(f);

    return 0;
}
```

Sequence Container Overview

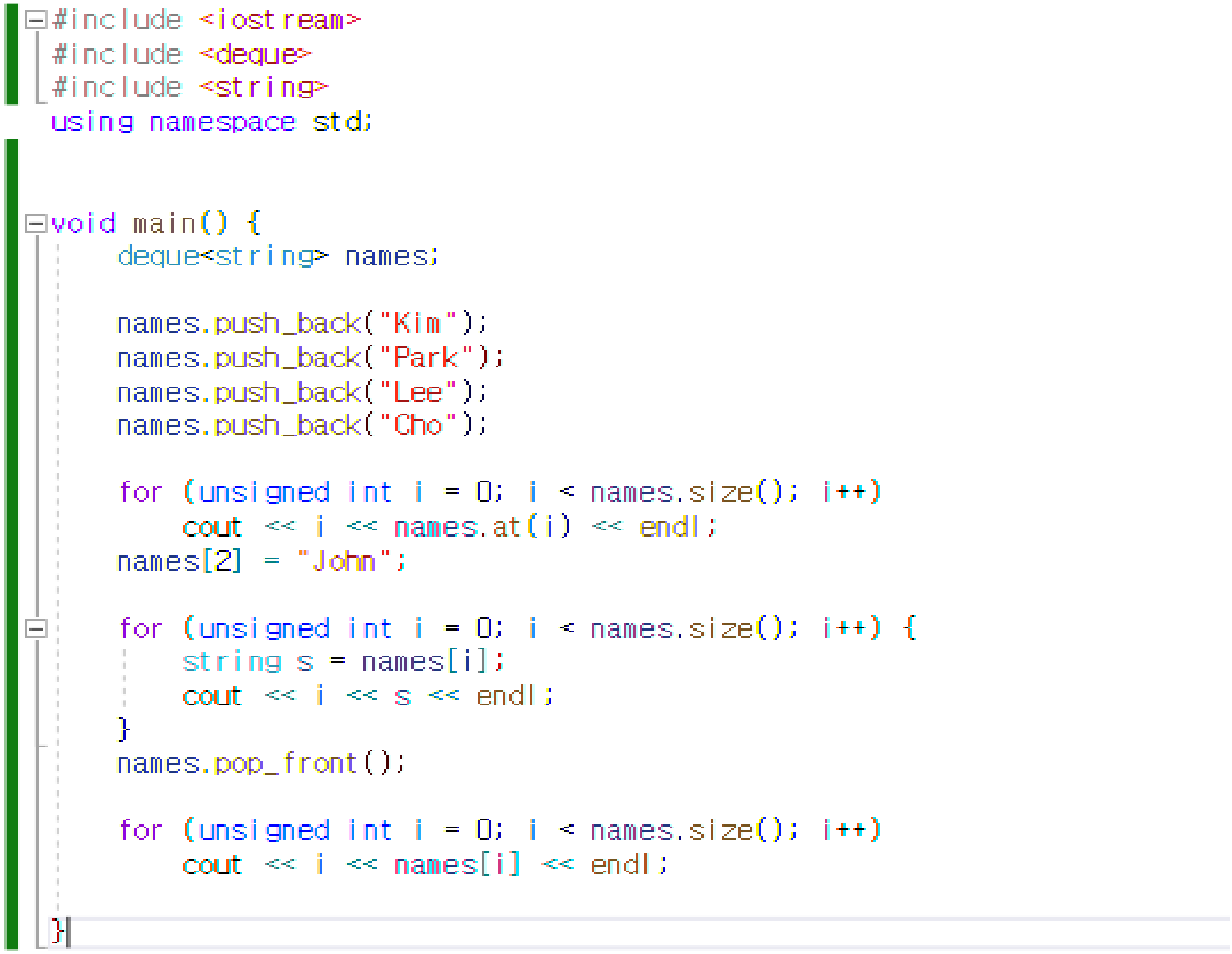
Library Name	Description	Example
`<vector>`	A dynamic array	STL-vector.cpp
`<list>`	randomly changing sequence of items	STL-list.cpp
`<stack>`	A sequence of items with pop and push at one end only (LIFO)	-
`<queue>`	A Sequence of items with pop and push at opposite ends (FIFO)	-
`<deque>`	Double Ended Queue with pop and push at both ends	STL-deque.cpp

Standard Template Library, List



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


void main() {
    list<string> names;

    names.push_back("Kim");
    names.push_back("Park");
    names.push_back("Lee");
    names.push_back("Cho");

    for (list<string>::iterator ai = names.begin(); ai != names.end();
        ai++)
        cout << *ai << endl;
    cout << endl;
    names.reverse();

    for (list<string>::iterator ai = names.begin(); ai != names.end();
        ai++)
        cout << *ai << endl;
}
```

Standard template library, Deque



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


void main() {
    deque<string> names;

    names.push_back("Kim");
    names.push_back("Park");
    names.push_back("Lee");
    names.push_back("Cho");

    for (unsigned int i = 0; i < names.size(); i++)
        cout << i << names.at(i) << endl;
    names[2] = "John";

    for (unsigned int i = 0; i < names.size(); i++) {
        string s = names[i];
        cout << i << s << endl;
    }
    names.pop_front();

    for (unsigned int i = 0; i < names.size(); i++)
        cout << i << names[i] << endl;

}
```