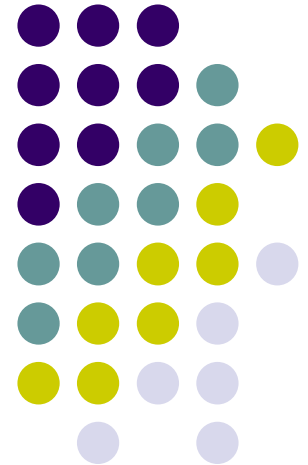


# File Structures

## 05. Appendix- Template

2020. Spring  
Instructor: Joonho Kwon  
jhkwon@pusan.ac.kr  
Data Science Lab @ PNU



# Template Non-types



- You can use non-types (constant values) in a template:

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

// return pointer to new N-element heap array filled with val
// (not entirely realistic, but shows what's possible)
template <typename T, int N>
T* valarray(const T &val) {
    T* a = new T[N];
    for (int i = 0; i < N; ++i)
        a[i] = val;
    return a;
}

int main(int argc, char **argv) {
    int *ip = valarray<int, 10>(17);
    string *sp = valarray<string, 10>("hello");
    ...
}
```

# What's Going On?



- The compiler **doesn't generate any code when it sees the template function**
  - It doesn't know what code to generate yet, since it doesn't know what types are involved
- When the compiler sees the function being used, then it understands what types are involved
  - It generates the **instantiation** of the template and compiles it (kind of like macro expansion)
    - The compiler generates template instantiations for *each* type used as a template parameter

# This Creates a Problem



```
#ifndef _COMPARE_H_
#define _COMPARE_H_

template <typename T>
int comp(const T& a, const T& b);

#endif // _COMPARE_H_
```

compare.h

```
#include <iostream>
#include "compare.h"

using namespace std;

int main(int argc, char **argv) {
    cout << comp<int>(10, 20);
    cout << endl;
    return EXIT_SUCCESS;
}
```

```
#include "compare.h"

template <typename T>
int comp(const T& a, const T& b) {
    if (a < b) return -1;
    if (b < a) return 1;
    return 0;
}
```

compare.cc

# Solution #1 (Google Style Guide prefers)



```
#ifndef _COMPARE_H_
#define _COMPARE_H_

template <typename T>
int comp(const T& a, const T& b) {
    if (a < b) return -1;
    if (b < a) return 1;
    return 0;
}

#endif // _COMPARE_H_
```

compare.h

```
#include <iostream>
#include "compare.h"

using namespace std;

int main(int argc, char **argv) {
    cout << comp<int>(10, 20);
    cout << endl;
    return EXIT_SUCCESS;
}
```

main.cc

# Solution #2 (you'll see this sometimes)



```
#ifndef __COMPARE_H__
#define __COMPARE_H__

template <typename T>
int comp(const T& a, const T& b);

#include "compare.cc"

#endif // __COMPARE_H__
```

compare.h

```
#include <iostream>
#include "compare.h"

using namespace std;

int main(int argc, char **argv) {
    cout << comp<int>(10, 20);
    cout << endl;
    return EXIT_SUCCESS;
}
```

main.cc

```
template <typename T>
int comp(const T& a, const T& b) {
    if (a < b) return -1;
    if (b < a) return 1;
    return 0;
}
```

compare.cc

# Q&A

