CS310

“Advanced C++: Templates and Generic Programming”

Nontype template parameters

Hüseyin Akcan
huseyin.akcan@ieu.edu.tr
Nontype template parameters

- For function and class templates, template parameters don't have to be types. They can also be ordinary values.
- When using such a template, you have to specify this value explicitly. The resulting code then gets instantiated.
You can also implement a stack by using a fixed-size array for the elements.

An advantage of this method is that the memory management overhead is avoided.

However, determining the best size for such a stack can be challenging.

A good solution is to let the user of the stack specify the size of the array as the maximum size needed for stack elements.

To do this, define the size as a template parameter.
Example:

- Define MAXSIZE as a nontype template parameter
- The new second template parameter, MAXSIZE, is of type int. It specifies the size of the array of stack elements:

```cpp
template <typename T, int MAXSIZE>
class Stack {
    private:
        T elems[MAXSIZE]; // elements.
    ...
};
```

- In addition, it is used in `push()` to check whether the stack is full:

```cpp
template <typename T, int MAXSIZE>
void Stack<T,MAXSIZE>::push (T const& elem)
{
    if (numElems == MAXSIZE) {
        throw "Stack<>::push(): stack is full";
    }
    elems[numElems] = elem; // append element.
    ++numElems; // increment number of elements.
}
```
**Example**

- To use this class template you have to specify both the element type and the maximum size:

  ```cpp
  Stack<int, 20> int20Stack;  // stack of up to 20 ints.
  Stack<int, 40> int40Stack;  // stack of up to 40 ints.
  Stack<std::string, 40> stringStack;  // stack of up to 40 strings
  ```

- Note that each template instantiation is its own type. Thus, `<int20Stack>` and `<int40Stack>` are two different types, and no implicit or explicit type conversion between them is defined. Thus, one cannot be used instead of the other, and you cannot assign one to the other.

- Default values for the template parameters can be specified:

  ```cpp
  template <typename T = int, int MAXSIZE = 100>
  class Stack {
      ...
  };
  ```

See stack4.hpp & stack4test.cpp
Nontype function templates

• You can also define nontype parameters for function templates. For example, the following function template defines a group of functions for which a certain value can be added:

```cpp
template <typename T, int VAL>
T addValue (T const& x)
{
    return x + VAL;
}
```
Restrictions for Nontype Template Parameters

- Note that nontype template parameters carry some restrictions. They cannot be:
  - Floating point values
  - Constant integral values (enumerations)
  - Strings
  - Pointers to objects with internal linkage