2020/06/02_数据结构_第3课_双向链表封装模板、C++迭代器

笔记本: 数据结构
创建时间: 2020/6/3 星期三 9:13
作者: ileemi
标签: C++迭代器, 双向链表封装模板

课前作业

类的内部有自定义类型，通过模板在类外使用的时候需要在定义前添加 typename 关键字。
typename 作用域<T>：：自定义类型

要是不加typename 编译器不知道::后面是调用静态变量还是一个类型

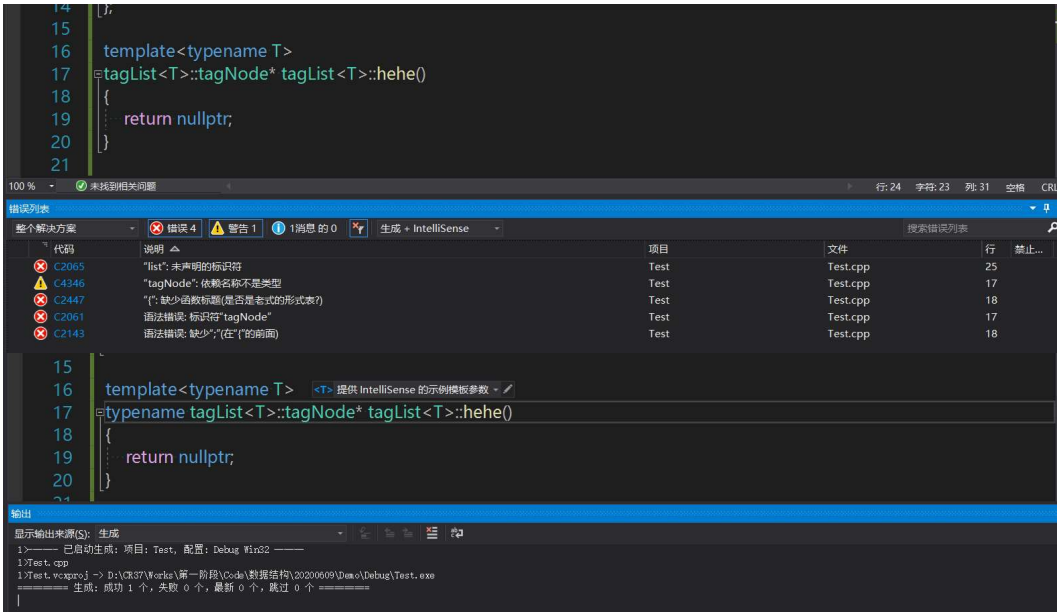
示例:

```
#include <iostream>
template<typename T>
struct tagList
{
    struct tagNode
    {
        T val;
    };
    tagNode* hehe();
    static tagNode* iii;
};

template<typename T>
typename tagList<T>::tagNode* tagList<T>::hehe()
{
    return nullptr;
}

int main()
{
    tagList<int> list;
```

```
    list.hehe();
}
```



C++STL 中的迭代器的简单使用

示例代码:

```
#include <iostream>
#include <list>
using namespace std;
int main()
{
    list<int> lst;
    lst.push_back(1);
    lst.push_back(2);
    lst.push_back(3);
    lst.push_front(999);
    lst.push_front(888);
    lst.push_front(666);

    // iterator 属于 list 内置类型
    list<int>::iterator itrBegin = lst.begin(); // 获取头部迭代器，指向头节点

    // 插入元素，此时插入结点时，结点并没有插入到头节点之前
    // 而是插入到了第一次 记录的头结点的位置之前
    //lst.insert(itrBegin, 10);
    //lst.insert(itrBegin, 11);
    //lst.insert(itrBegin, 12);
    //lst.insert(itrBegin, 13);
```

```
    //lst.insert(itrBegin, 15);

    // 每次插入数据，都插入在结点之前
    lst.insert(itrBegin, 10);
    lst.insert(lst.begin(), 10);
    lst.insert(lst.begin(), 11);
    lst.insert(lst.begin(), 12);
    lst.insert(lst.begin(), 13);


    // 从头部指针位置开始，从容器擦除指定的元素
    lst.erase(lst.begin());
    lst.erase(lst.begin());
    lst.erase(lst.begin());
    lst.erase(lst.begin());

    // 提供移动位置
    auto itr = lst.begin();
    itr++;  // 移动到后继结点，支持++、--
    *itr = 999; // 修改后继结点的数据

    auto itr1 = lst.end();  // 指向尾结点
    //*itr1 = 999;  // 访问位置上的数值不做修改，出发断言

    itr1--;
    *itr1 = 999;

    // 遍历链表中每个结点对应的数据值
    for (auto itr = lst.begin(); itr != lst.end(); itr++)
    {
        cout << *itr << endl;
    }
    cout << endl;
    //lst[2] = 777;
    // 简化for循环条件，遍历链表中每个结点对应的数据值
    // 只要实现了迭代器，C++允许下面这样的写法
    for (auto val:lst)
    {
        cout << val << endl;
    }
    return 0;
}
```

C++ 迭代器

简要概述：迭代器是一种检查容器内部元素并遍历元素的数据类型。

扩展

> Iterator（迭代器）模式又称Cursor（游标）模式，用于提供一种方法顺序访问一个聚合对象中各个元素，同时不暴露对该对象内部的表示。
> 或者可以这样理解：Iterator 模式是运用于聚合对象的一种模式，通过运用该模式，使得我们可以在不知道对象内部表示的情况下，按照一定的顺序（由Iterator提供的内部方法）访问聚合对象中的各个元素。

实现 Iterator 模式的两种方式：内嵌类和友元类。

> 参考：c++迭代器（iterator）详解