深蓝学院
shenlanxueyuan.com

C++ 基础与深度解析
Project5-元编程 思路提示
2022-08-01

主讲人 天哲

纲要

- 基本设计
- 翻转的实现
- 加法实现
- 除法实现
- 扩展内容
- 其他

基本设计

- 基本的参数包
- 参数包和折叠表达式

```
template <unsigned int... ints>
struct Cont {};

//print all ints in Cont
template <unsigned int... ints>
void print(Cont<ints...>)
{
    ((std::cout << ints << ' '), ...);
    std::cout << std::endl;
}
```

基本设计-简单递归，去0

- 递归：每次去除头部的0
- 递归终止条件：0被去除完了，或者是0本身

```
//cut leading zero from Cont<>,when there is no leading zeros,
keep the same
template <typename T>
struct CutHeadZeros
{
    using result = T;
};
//if there is leading zero, cut one at a time
template <unsigned int... tail>
struct CutHeadZeros<Cont<0, tail...>>
{
    using result =  CutHeadZeros<Cont<tail...>>::result;
};
//if result is Cont<0>
template <>
struct CutHeadZeros<Cont<0>>
{
    using result = Cont<0>;
};
```

➢翻转的实现

- 递归：每次把第一个数字放到处理完的尾部
- 递归终止条件：所有数字都被处理完

```
//Reverse Cont
template <typename = Cont<>, typename = Cont<>>
struct Reverse;

//like a stack, first in is at tail
template <unsigned int first, unsigned int... remain, unsigned int... done>
struct Reverse<Cont<first, remain...>, Cont<done...>> {
    using result =  Reverse<Cont<remain...>, Cont<first, done...>>::result;
};

//End when all are done
template <unsigned int... done>
struct Reverse<Cont<>, Cont<done...>> {
    using result = Cont<done...>;
};
```

➢加法的实现

```
//Add Cont from first to last ,Cont<> should already be reversed
template <typename num1 = Cont<>, typename num2 = Cont<>, unsigned int carry = 0, typename sum = Cont<>>
struct AddImp;

//alias template of Add<Cont<>,Cont<>>
template <typename T1 = Cont<>, typename T2 = Cont<>>
using Add = CutHeadZeros<typename AddImp<typename Reverse<T1>::result,
typename Reverse<T2>::result>::result>::result;
```

- AddImp从左到右分别是 第一个数，第二个数，进位和结果
- Add则是包括各种处理，例如翻转，去0等就不再赘述

➢加法的实现

```
//main of AddImp, take the first of each and put them into result, and removed from original
template <unsigned int first1, unsigned int first2, unsigned int carry, unsigned int... num1, unsigned int... num2, unsigned int... ret>
struct AddImp<Cont<first1, num1...>, Cont<first2, num2...>, carry, Cont<ret...>>
{
    static_assert(first1 < 10 && first2 < 10 && carry < 2, "Eash digit should be smaller than 10");
    using result =  AddImp<Cont<num1...>, Cont<num2...>, (first1 + first2 + carry) / 10, Cont<ret..., (first1 + first2 + carry) % 
10>>::result;
};
```

- 递归处理：每次处理1位，取余之后放入结果位，同时考察是否进位
- 终止条件：一个数被计算完毕/全部计算完毕

```
//if second Cont<> is shorter
template <unsigned int first1, unsigned int carry, unsigned int... num1, unsigned int... ret>
struct AddImp<Cont<first1, num1...>, Cont<>, carry, Cont<ret...>>
{
    using result =  AddImp<Cont<num1...>, Cont<>, (first1 + carry) / 10, Cont<ret..., (first1 + carry) % 10>>::result;
};

//if first Cont<> is shorter
//other code here...
//end recuresive, both Cont ends, calculate last time carry
template <unsigned int carry, unsigned int... ret>
struct AddImp<Cont<>, Cont<>, carry, Cont<ret...>>
{
    using reversedType = std::conditional<(carry > 0), Cont<ret..., carry>, Cont<ret...>>::type;
    using result = Reverse<reversedType>::result;
};
```

除法的实现

```
//BigNum Divid by using long-division
template <typename input = Cont<>, typename quotien = Cont<>, unsigned int base = 10, unsigned int divisor = 10, unsigned remainder = 0>
struct BigNumDiv;

//Each time , extract one head digit from divident and calculate corresponding quotient and remain
template <unsigned int base, unsigned int divisor, unsigned remainder,
          unsigned int head, unsigned int... dividend, unsigned int... quotient>
struct BigNumDiv<Cont<head, dividend...>, Cont<quotient...>, base, divisor, remainder>

{
    static_assert(base > 1 && divisor > 0, "Base should be greater then 1 and divisor should greater than 0");
    constexpr static unsigned int sdivident = remainder * base + head;
    constexpr static unsigned int digit = sdivident / divisor;
    using result = typename BigNumDiv<Cont<dividend...>, Cont<quotient..., digit>, base, divisor, sdivident % divisor>::result;
    constexpr static unsigned int remain = BigNumDiv<Cont<dividend...>, Cont<quotient..., digit>, base, divisor, sdivident % divisor>::remain;
};
```

- 递归处理：每次处理1位，把每一位的结果传到下一个递归
- 终止条件：被除数全部被处理完毕，得到最后的商和余数

```
//recursion stopper when all digits in dividend is processed
template <unsigned int base, unsigned int divisor, unsigned remainder, unsigned int... quotient>
struct BigNumDiv<Cont<>, Cont<quotient...>, base, divisor, remainder>

{
    //using result = Cont<quotient...>;
    using result = CutHeadZeros<Cont<quotient...>>::result;
    constexpr static unsigned int remain = remainder;
};
```

➢进制转换的实现

```cpp
template <typename T = Cont<>, unsigned int base = 10, unsigned int target = 10, typename = Cont<>>
struct BaseConversion;

template <unsigned int base, unsigned int target, unsigned int... input, unsigned int... processed>
struct BaseConversion<Cont<input...>, base, target, Cont<processed...>>
{
    static_assert(base < 11 && target < 11 && base > 1 && target > 1, "Eash digit should be smaller than 10");
    constexpr static unsigned int remain = BigNumDiv<Cont<input...>, Cont<>, base, target>::remain;
    using tempResult = BigNumDiv<Cont<input...>, Cont<>, base, target>::result;
    using result =  BaseConversion<tempResult, base, target, Cont<remain, processed...>>::result;
};
```

- 递归处理：保存每次得到的商和余数，把余数放到结果中
- 终止条件：所有数位全部处理完毕，商为0

```cpp
template <unsigned int base, unsigned int target, unsigned int... processed>
struct BaseConversion<Cont<0>, base, target, Cont<processed...>>
{
    //when the first input = Cont<0>□return Cont<0>
    using result = std::conditional<(sizeof...(processed) > 0), Cont<processed...>, Cont<0>>::type;
};
```

➢其他提示

- 元编程中的循环大多需要靠递归完成，重点是要思考1.怎么递归 2.递归改变/终止的条件
- 善用constexpr，可以用数学计算的不妨用数学计算解决…例如加法中去余操作是可以用%运算符的
- 可以考虑使用static_assert或者concept做一些检查

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening