东南大学吴健雄学院
CHIEN-SHIUNG WU COLLEGE OF SEU

C++程序设计

——模板与数据结构

内容

模板的定义与使用

顺序表模板

类模板推演中一个重要问题（模板与类参数）

函数指针*

模板的定义与使用

模板是什么？ —— 通用的样式

函数模板 —— 类型无关的通用函数

```
int max(int , int , int );
Complex max(Complex , Complex , Complex );
string max(string , string , string);
```

函数模板：类型参数化（可变）

```
template <typename T>
T  max(T, T, T) ;
```

怎么写函数模板？ 例如：求数组元素最大值的函数模板

step 1: 找个最简单类型做特例

```
int max(int a[ ], int n);
```

step 2: 类型参数化

```
template<typename T>
T  max( T a[ ], int n );
```

模板的使用 —— 模板实例化（template instantiation）

```
int a[5];

string st[10];

Complex c[20];

//数据输入

int imax = max(a, 5);                    //imax = max<int> (a, 5);

string  smax = max(st, 10);              //smax = max<string> (a, 10);

Complex cmax = max(c, 20);

//直接用实参调用
```

```
template<typename T>

T  max(T [ ], int );
```

【例6.2】* 函数模板在矩阵操作中的应用（输入、输出、转置）。

```
template<typename T>
void input(T* p, int m, int n);          //矩阵输入，指针为数组中数据的类型

template<typename T>
void output(T* p, int m, int n);         //矩阵输出

template<typename T>
void trans(T* p1, T* p2,  int m, int n);  //矩阵转置
```

```
int main()
{    int  a[2][3], ta[3][2];
     input <int> ((int*)a, 2, 3);
     output <int> ((int*)a, 2, 3);
     cout << endl;
     //trans如何调用?
     output <int> ((int*)ta, 3, 2);
     cout << endl;
}
```

trans <int> ((int*)a,(int*) ta, 2, 3);

```
template<typename T>
void input(T* p, int m, int n)
{   int i, j;
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
            cin >> *(p + n * i + j);
}

template<typename T>
void trans(T* p1, T* p2, int m, int n)
//原数组m行n列，转置数组n行m列
{   int i, j;
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
            *(p2 + m * j + i)= *(p1 + n * i + j);
}
```

```
template<typename T>
void output(T* p, int m, int n)
{   int i, j;
    for(i = 0; i < m; i++)
    {   for(j = 0; j < n; j++)
            cout << setw(4) << *(p + n * i + j);
        cout << endl;
    }
}
```

可以这样使用整型数组对象吗?

```
Array  a;          //整型数组对象
int m;
for( int i = 0; i < 10; i++)
{    cin >> m;
      a.insert(m, i);
}   //读入数据
a.sort();          //排序
cout << a;         //输出数组
```

封装整型数组类

```
class Array
{     int  arr [100] ;
      int  last;        //最后一个元素下标
public:
     array() { last = -1; }
     void insert( int , int );    //某位置插入元素
     friend ostream& operator <<(ostream&, const
Array&) ;
     void sort();
     //其他成员暂略
};
```

类模板 —— 某个类型无关的通用类

数组类

```
template< typename T, int n>
class Array
{     T   arr [  n  ] ;
      int  last;        //最后一个元素下标
public:
     array() { last = -1; }
     void insert(  T   , int );    //某位置插入元素
     friend ostream& operator <<(ostream&,
const Array&) ;
     void sort();
     //其他成员暂略
};
```

T —— 模板类型参数

n —— 模板非类型参数
是一个潜在常量

模板推演（实例化）

```
Array <int, 100> iarray;
Array < string, 10> sarray;
Array <Complex, 20> carray;
```

怎么写类模板？ 2步走

类模板中成员函数均为函数模板 —— 类型参数化

```
template< typename T, int n>
void Array<T, n> :: print()              //输出数组元素
{    for(int i = 0; i <= last; i++)  cout << arr[i] << " ";    }

template<typename T,int n>
void Array<T, n> :: insert( T m , int pos )    //m插入在下标 pos 处
{   if(pos < 0)  pos = 0;                  //健壮性处理
    if(pos > last) pos = last + 1;         //健壮性处理
    for (int i = last; i >= pos; i--)
        arr[i+1] = arr[i];                 //逐个移动元素
    arr[pos] = m;                          //m放在下标pos处
    last++;
}
```

顺序表模板与数据结构

什么是顺序表？ —— 元素连续存储（数组）

顺序表的封装 —— 数组类的封装

```
template <typename T, int size>
class Seqlist
{    T list [size];        //数组
     //int maxsize;    //数组容量，教材例题用，非必需
     int last;             //最后元素下标
public:
     Seqlist()             //通常初始化为空表
    {   last = -1;   }  //最后元素下标 -1， 表示没有元素，空表特征
// to be continued……
```

```
// continued……
public:
    Seqlist();
    int Length() const   {   return last + 1;   }      //计算表长
    bool IsEmpty() { return last == -1;}               //判断表是否空，没有元素
    bool IsFull() {   return last == size -1;   }      //判断表是否满，数组空间占满
    friend ostream& operator <<(ostream&, const Array&)；
    T& operator[ ](int i)；                            //下标运算，带下标合法性检查
    bool Remove(T& key);                               //删除元素key，返回是否正常删除
    bool  Remove(int pos);                             //删除下标pos处的元素
    bool Insert(T& x, int pos);                        //x插入在下标pos处，返回是否正常插入
    // bool Insert(T&, int) 可用于建立数组
    bool InsertAsc( T& x);                             //插入x，保持升序，可用于建立升序数组
    void InsertSort( );                                //插入排序
    int Search(T& key) const;                          //查找key，返回下标
    int AscBisearch(T& key) const;                     //升序数组二分查找
};
```

```
Template <typename T, int size>
bool Seqlist<T, size> :: Remove (T& key)      //删除关键字数据
{    if(last == -1) {    cout << "空数组无可删！" << endl;      return false;    }
     int pos = Search(key);
     return Remove(pos);
}

Template <typename T, int size>
bool Seqlist<T, size> :: Remove (int pos)     //删除下标元素
{    if (last == -1) {    cout << "空数组无可删！" << endl;      return false;     }
     if (pos < 0 || pos > last)
     {    cout << "待删元素不在数组内！" << endl;      return false;     }
     for (int i = pos; i < last; i++)    arr[i] = arr[i + 1];
     last--;
     return true;
}
```

```
Template <typename T, int size>
bool Seqlist<T, size> :: InsertAsc (T& key)     //插入元素，保持升序
{    if(last +1 == size) {    cout << "数组已满，插入失败！" << endl;    return false;   }
     for(int i = last; i >= 0; i--)     //找位置并挪位
          if(key < arr[i])   arr[i+1] = arr[i ];
          else break;
     arr[i + 1] = key;
      last++;
      return true;
}
```

```
Template <typename T, int size>
void Seqlist<T, size> :: InsertSort ()      //插入排序法（升序）
{    for(int i = 1; i <= last; i++)     //待插入元素下标，即排序次数
     {    T enter = arr[i];
          for(int j = i – 1; j > = 0; j--)
               if(enter < arr[j])    arr[j+1] =  arr[j];
               else break;
          arr[j + 1] = enter;
     }
```

查找算法：顺序，二分；

哈希（散列），二叉树等。

排序算法：选择，冒泡；

插入，归并；

希尔排序、快排序等；

关于函数返回引用

```
Template <typename T, int size>
T& Seqlist<T, size> :: operator[ ] (int i);

Seqlist<int, 10>  a;
cout  << a[3];    √
a[3] = 5;         √
```

```
Template <typename T, int size>
T   Seqlist<T, size> :: operator[ ] (int i);

Seqlist<int, 10>  a;
cout  << a[3];    √
a[3] = 5;         ×
```

返回值：　函数中创建临时变量，传递值后即撤销；

返回引用：函数中不创建临时变量，被返回变量的生命期须超过函数生命期。

从现象看：返回值获得的仅仅是变量的值，返回引用获得的是变量。

用类模板创建对象 —— 类模板的推演 / 类模板的实例化

```
Seqlist <int, 20>  iArray, ascArray;   //创建整数空表
int x;
for(int i = 0; i < 10; i++)   //读入10个数据
{       cin >> x;
        iArray.Insert(x, i);          //无序表
        ascArray.InsertAsc(x);   //升序表
}
iArray.InsertSort();                  //升序排序
cin >> x;
if (iArray.Remove(x) )          //成功删除元素
      iArray.print() ;             //输出数组
```

```
Seqlist <Fraction, 50>  fArray;  //分数空表
Fraction f;
for(int i = 0; i < 10; i++)
{
        int fzz, fmm;                 //读入10个分数
        cin >> fzz >> fmm;
        f.set(fzz, fmm);
        fArray.Insert(f, i);      //无序表

}
```

上节内容回顾

模板

- 概念：通用、有条件（相对，非绝对）
- **函数模板**：形参类型通用（参数化）
- **类模板**-顺序表模板类：线性表、顺序表概念，元素类型通用、表容量通用（模板类型参数、模板非类型参数）
- **如何构造模板：**从特殊（最简单实例）到一般（参数化）
- 如何用模板构造各种特殊的类实例：从一般到更广泛的特殊 ？？？

返回类型为引用

插入排序、归并排序算法

索引查找、索引排序与指针数组 *

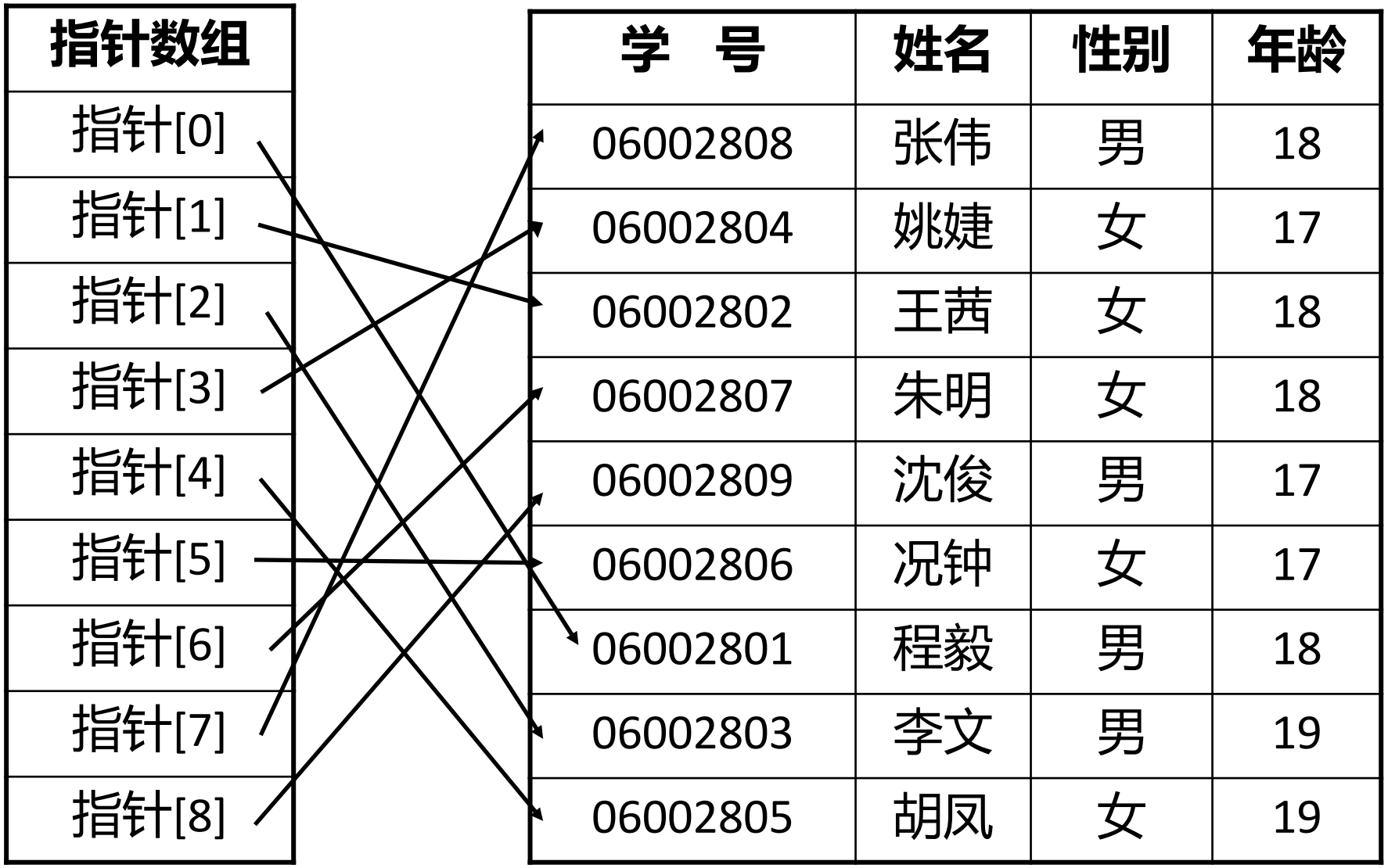
索引查找（Index Search）及排序用于

- 复杂数据对象或大范围内的数据查找及排序
- 避免排序时大量数据移动

通过索引（目录）实现 —— 指针数组

索引

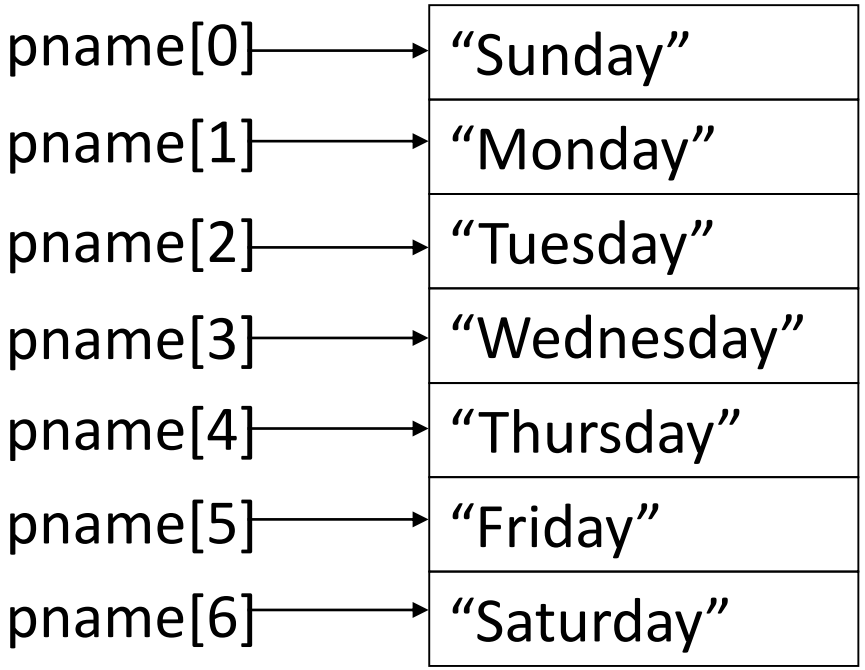
数据 class Person { };
Person per [10];



指针数组
指针[0]
指针[1]
指针[2]
指针[3]
指针[4]
指针[5]
指针[6]
指针[7]
指针[8]

学　号	姓名	性别	年龄
06002808	张伟	男	18
06002804	姚婕	女	17
06002802	王茜	女	18
06002807	朱明	女	18
06002809	沈俊	男	17
06002806	况钟	女	17
06002801	程毅	男	18
06002803	李文	男	19
06002805	胡凤	女	19

例：索引排序，用指针数组作为索引操作字符串数组。



Person per[10], *pstr[10];

数据类型扩展... ...

```
string weekday[7];    //或char weekday[7][20];
string* pstr[7];      //或char* ppstr[7];
/* 建立初始索引 */
for(i = 0; i < 7; i++)   ppstr[i] = &weekday[i];
//或for(i = 0; i < 7; i++)   ppstr[i] = weekday[i];
/* 对索引排序 */
for(i = 0; i < 7; i++)                    //冒泡
    for(j = 6; j > i; j--)
        if(pstr[j] < pstr[j-1])
        {   string* t = pstr[j];    //交换指向
            pstr[j] = pstr[j-1];
            pstr[j-1] = t;
        }
for(i = 0; i < 7; i++)
    cout << *pstr[i] << “ “;    //或cout << ppstr[i]
```

类模板推演中一个重要问题（模板与类参数）

类模板用最简单数据类型抽象而成，实例化为对象时会不会有问题？例如：

```
Template <typename T, int size>
void Seqlist<T, size> :: InsertSort ()
{    for(int i = 1; i <= last; i++)
{     T enter = arr[i];
          for(int j = i – 1; j > = 0; j--)
              if(enter < arr[j])    arr[j+1] =  arr[j];
              else break;
         arr[j + 1] = enter;
}
```

```
class Complex
{     double re, im;
       double r;      //模
public:                 //其他成员略
       int operator < (Complex c)
       {    return (r < c.r);    }
};

Seqlist <Complex, 10>  cArray;
 cArray.InsertSort();   //按模值
```

结论：实例化类要重载相关运算符：> < == != << >> … …
若含指针成员还要重载 =

定积分问题的模板之一 —— 类模板（被积函数对象作为数据成员，类型参数化）

```
template<typename T>
class Integral
{    double a, b, step;  //上下界、步长
     int n;              //分区数
     double result;      //积分值
     T cf;               //被积函数对象
public:
    Integral(double =0, double =0, int =1000);
    void putlimits(double, double, int);
    double calculate();  //梯形算法
    void print();
};
```

```
template<typename T>
double Intigral<T>:: calculate()
{     result = (cf.fun(a) + cf.fun(b)) / 2;
      for(int i = 1; i < n; i++)
          result += cf.fun(a + step * i);
      result *= step;
      return result;
}
Integral<F1> intg1(0, 5, 1000);
cout << intg1.calculate() << endl;
Integrate<F2> intg2(0, 5, 2000);
cout << intg2.calculate() << endl;
```

```
template<typename T>
double Intigral<T>:: integrate()
{     result = (cf.fun(a) + cf.fun(b)) / 2;
       for(int i = 1; i < n; i++)
            result += cf.fun(a + step * i);
      result *= step;
      return result;
}

Integral<F1> intg1(0, 5, 1000);
cout << intg1.integrate() << endl;
Integral<F2> intg2(0, 5, 2000);
cout << intg2.integrate() << endl;
```

```
class F1{    //函数封装为对象
public:
     double fun(double x)
     {   return x * x * x;   }
};

class F2{
public:
     double fun(double x)
     {   return x * x + sin(x);   }
};
```

定积分问题的模板 之二—— 函数模板（被积函数对象作为参数，类型参数化）

```
template<typename T>
double integral (T cf, double a, dobule b, int n=1000)
{   double result, step;
    result = (cf.fun(a) + cf.fun(b))/2;
    step = (b - a) / n;
    for (int i= 1; i < n; i++)
        result += cf.fun(a + i * step);
    result *= step;
    return result;
}
```

```
/* 函数类定义同上例 */
class F1{  };
class F2{  };

F1 f1;
F2 f2;
double intgf1, intgf2;
intgf1 = integral(f1, 0, 3, 1000);
intgf2 = integral(f2, 3, 10, 10000);
```

函数指针 *

函数名是什么？ 该函数代码的入口地址（指针常量）

函数指针做什么用？ 可以指向函数的入口地址，像函数名一样使用

函数指针的定义 —— 具有所指函数的类型，例如：

```
double (*pf) (double);          //指向f(x)的指针
int  (*pfi) (double, double);   //指向f(x,y) 的指针
```

若有函数： double fun (double x) { return x * x * x – 5 * x + 7; }

```
pf = fun;
```

则可用pf调用函数： cout << (*pf)(3.5) << endl; **这有什么意义?**

定积分问题非模板通用函数 —— 被积函数用指针表达

```
double integral (double (*pf)（double）, double a, double b, int n = 1000)
{    double result, step;
     result = ((*pf)(a) + ((*pf(b))/2;
     step = (b - a) / n;
     for (int i= 1; i < n; i++)
         result += (*pf) (a + i * step);
     result *= step;
     return result;
}
```

调用方法 —— 实参传递函数名

```
/* sinx的定积分 */
double  y = integral(sin, 0, 3.14 / 2);

/* 自定义f1(x)的定积分 */
double  f1(double  x)
{   return x * x + 3 * x + 1;   }

y = integral(f1, 1, 4, 2000);
```