Module04-08
C++ 标准库：数值

- 数据结构简介
- 标准容器
- 常用算法简介
- 标准算法与函数对象
- 迭代器
- 字符串
- I/O 流
- 数值

标准库 - 数值

- 数值 (Numeric)
  - 标准数学函数
  - 向量算术
  - 复数算术
  - 通用数值算法
  - 随机数

数值 - 标准数学函数

- 标准数学函数
  - 下列函数在 <cmath> 中

函数接口	说明
double abs(double)	绝对值
double fabs(double)	绝对值
double ceil(double d)	返回不小于 d 的最小整数
double floor(double d)	返回不大于 d 的最大整数
double sqrt(double d)	d 的平方根，d 必须非负数
double pow(double d, double e)	d 的 e 次幂
double pow(double d, int i)	d 的 i 次幂
double cos(double)	余弦
double sin(double)	正弦
double tan(double)	正切
double acos(double)	反余弦

数值 - 标准数学函数

- 标准数学函数（续）

函数接口	说明
double asin(double)	反正弦
double atan(double)	反正切
double atan2(double x, double y)	atan(x/y)
double sinh(double)	双曲正弦
double cosh(double)	双曲余弦
double tanh(double)	双曲正切
double exp(double d)	指数，以 e 为底
double log(double d)	自然对数，以 e 为底， d 须大于 0
double log10(double d)	以 10 为底的对数， d 须大于 0
double modf(double d, double* p)	返回 d 的小数部分，整数部分存入 *p
double frexp(double d, int* p)	找出 [0.5~1.0) 中的 x 和 y ，使 d = x * pow(2, y) ，返回 x，并将 y 存入 *p
double fmod(double d, double m)	浮点数的余数，符号与 d 相同
double ldexp(double d, int i)	返回 d * pow(2, i)

数值 - 标准数学函数

- 标准数学函数（续）
  - 下列函数在 <cstdlib> 中

函数接口	说明
int abs(int)	绝对值
long abs(long)	绝对值
long labs(long)	绝对值
div_t div(int n, int d)	用 d 除 n，返回（商，余数）
ldiv_t div(long n, long d)	用 d 除 n，返回（商，余数）
ldiv_t ldiv(long n, long d)	用 d 除 n，返回（商，余数）

数值 - 向量算术

- 关于 valarray
  - valarray 是专注于向量算术的一种数据结构
  - valarray 类型是由 4 个用于刻画 valarray 中各个部分的辅助类型支持的：
    - slice_array 和 gslice_array 描述切割概念
    - mask_array 通过屏蔽各个元素的进和出来刻画数据的子集
    - indirect_array 列出需要考虑的元素的下标

数值 - 向量算术

- 创建 valarray
  - 示例：

```
std::valarray<int> va1(10); // 10个元素，初始值为0
std::valarray<float> va2(5.7, 10); // 10个元素，初始值为5.7

// 通过数组创建
int a[] = { 3, 6, 18, 3, 22 };
std::valarray<int> va3(a, sizeof(a) / sizeof(a[0]));
std::valarray<int> va4(a + 1, 3);
```

- valarray 的操作
  - 示例：

```
valarray<int> va(10);
// 将va中所有元素赋值为4
va = 4;

// 将va中所有元素乘以5
va *= 5;

// va2和va3对应下标的元素相乘
// va2[0] * va3[0] .. va2[size-1] * va3[size-1]
va1 = va2 * va3;

valarray<int> va0(2, 10);
va0 *= 5.9; // Error, 类型不匹配

int m = va.max();    // 返回va中最大元素
int sm = va.sum();   // 返回va中元素的总和
```

数值 - 向量算术

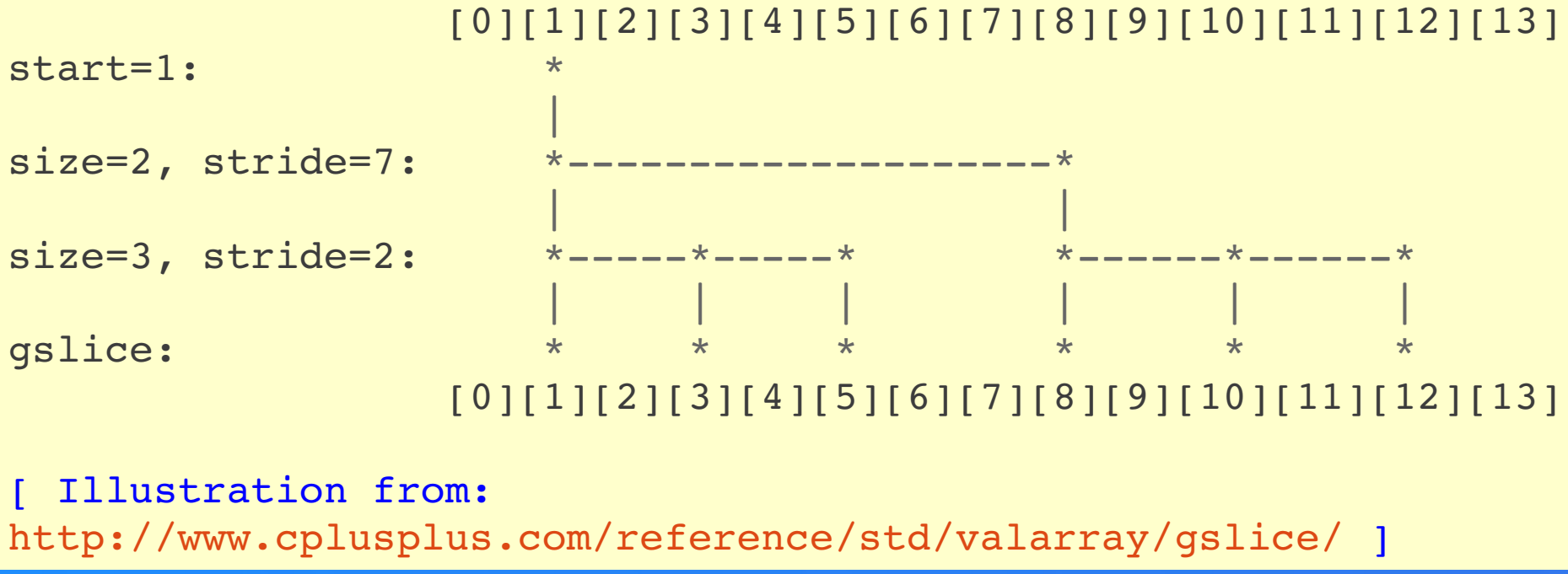
- slice_array 和 mask_array
  - 示例：

```
valarray<int> va(16);

// 返回子集：从va下标为2的元素开始，跨距为3，共取4个元素
// 即 va[2], va[5], va[8], va[11] 4个元素
va[slice(2, 4, 3)];
va[slice(2, 4, 3)] = 8; // 将上述4个元素值置为8


// 返回子集：va中大于7的元素
va[va > 12]; // mask_array
bool ba[] = { true, false, false, true, false, true };
valarray<bool> mask(ba, 6);
valarray<int> va2 = va[mask];
```

数值 - 向量算术

- gslice_array
  - gslice 切片：

```
size_t len[] = { 2, 3 };
size_t str[] = { 7, 2 };
valarray<size_t> length(len, 2);
valarray<size_t> stride(str, 2);
gslice slc(1, length, stride);
```



[Illustration from:
http://www.cplusplus.com/reference/std/valarray/gslice/]

- gslice_array
  - 示例：

```
// From book: C++ Standard Library (Nicolai Josuttis)
template<class T>
void printValarray3D(const valarray<T>& va, int dim1, int dim2) {
    for (int i = 0; i < va.size() / (dim1 * dim2); ++i) {
        for (int j = 0; j < dim2; ++j) {
            for (int k = 0; k < dim1; ++k) {
                cout << va[i * dim1 * dim2 + j * dim1 + k]<< ' ';
            }
            cout << '\n';
        }
        cout << '\n';
    }
    cout << endl;
}
```

- gslice_array
  - 示例（续1）：

```
int main() {
    /* valarray with 24 elements
     * - two groups
     * - four rows
     * - three columns
     */
    valarray<double> va(24);
    // fill valarray with values
    for (int i = 0; i < 24; i++) {
        va[i] = i;
    }
    // print valarray
    printValarray3D(va, 3, 4);
  // we need two two-dimensional subsets of three times 3 values
    // in two 12-element arrays
    size_t lengthvalues[] = { 2, 3 };
    size_t stridevalues[] = { 12, 3 };
```

- gslice_array
  - 示例（续 2 ）：

```
    valarray<size_t> length(lengthvalues, 2);
    valarray<size_t> stride(stridevalues, 2);
    // assign the second column of the first three rows
    // to the first column of the first three rows
    va[gslice(0, length, stride)] =
            valarray<double> (va[gslice(1, length, stride)]);
    // add and assign the third of the first three rows
    // to the first of the first three rows
    va[gslice(0, length, stride)] +=
            valarray<double> (va[gslice(2, length, stride)]);
    // print valarray
    printValarray3D(va, 3, 4);
}
```

- indirect_array
  - 示例：

```
// From book: C++ Standard Library (Nicolai Josuttis)
template<class T>
void printValarray(const valarray<T>& va, int num) {
    for (int i = 0; i < va.size() / num; i++) {
        for (int j = 0; j < num; j++) {
            cout << va[i * num + j] << ' ';
        }
        cout << endl;
    }
    cout << endl;
}
```

- indirect_array
  - 示例（续）：

```
int main() {
    // create valarray for 12 elements
    valarray<double> va(12);
    // initialize valarray by values 1.01, 2.02, ... 12.12
    for (int i = 0; i < 12; i++) {
        va[i] = (i + 1) * 1.01;
    }
    printValarray(va, 4);
    /* create array of indexes
     * - note: element type has to be size_t
     */
    valarray<size_t> idx(4);
    idx[0] = 8; idx[1] = 0; idx[2] = 3; idx[3] = 7;
    // use array of indexes to print the ninth, first, fourth,
and eighth elements
    printValarray(valarray<double> (va[idx]), 4);
```

数值 - 向量算术

- indirect_array
  - 示例（续）：

```
    // change the first and fourth elements and print them again
indirectly
    va[0] = 11.11; va[3] = 44.44;
    printValarray(valarray<double> (va[idx]), 4);
    // now select the second, third, sixth, and ninth elements
    // and assign 99 to them
    idx[0] = 1; idx[1] = 2; idx[2] = 5; idx[3] = 8; va[idx] = 99;
    // print the whole valarray again
    printValarray(va, 4);
}
```

数值 - 复数算术

- 复数 (complex)
  - （略）：

数值 - 通用数值算法

- 通用数值算法

  以下函数在 <numeric> 中

  - accumulate()
    - 累加一个序列中所有元素的值
  - inner_product()
    - 求 2 个序列的内积 (dot)
  - partial_sum()
    - 返回一个序列的增量和的序列，如对序列 {12, 23, 8, 51} 操作后，返回 {12, 12+23, 12+23+8, 12+23+8+51}
  - adjacent_difference()
    - 返回一个序列的增量变化的序列，如对序列 {12,23,8,51} 操作后，返回 {12, 23-12, 8-23, 51-8}

- 通用数值算法

```
template<typename InputIterator, typename T>
T accumulate(InputIterator first, InputIterator last, T init);

template<typename InputIterator, typename T,
         typename BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init,
         BinaryOperation binary_op);

template<typename InputIterator1, typename InputIterator2,
         typename T>
T inner_product(InputIterator1 first1, InputIterator1 last1,
         InputIterator2 first2, T init);

template<typename InputIterator1, typename InputIterator2,
         typename T, typename BinaryOperation1,
         typename BinaryOperation2>
T inner_product(InputIterator1 first1, InputIterator1 last1,
         InputIterator2 first2, T init, BinaryOperation1
binary_op1, BinaryOperation2 binary_op2);
```

- 通用数值算法

```
template<typename InputIterator, typename OutputIterator>
OutputIterator partial_sum(InputIterator first,
    InputIterator last, OutputIterator result);

template<typename InputIterator, typename OutputIterator,
        typename BinaryOperation>
OutputIterator partial_sum(InputIterator first,
    InputIterator last, OutputIterator result,
    BinaryOperation binary_op);

template<typename InputIterator, typename OutputIterator>
OutputIterator adjacent_difference(InputIterator first,
    InputIterator last, OutputIterator result);

template<typename InputIterator, typename OutputIterator,
        typename BinaryOperation>
OutputIterator adjacent_difference(InputIterator first,
    InputIterator last, OutputIterator result,
    BinaryOperation binary_op);
```

数值 - 通用数值算法

- 示例

```
int a[] = { 12, 23, 8, 51 };
int b[] = { 2, 6, 11, 61 };
int sum = accumulate(a, a + 4, 0); // 94
cout << sum << endl;

int prod = inner_product(a, a + 4, b, 0); // 3361
cout << prod << endl;

int c[4];
partial_sum(a, a + 4, c); // 12 35 43 94
copy(c, c + 4, ostream_iterator<int> (cout, " "));
cout << endl;

adjacent_difference(a, a + 4, c); // 12 11 -15 43
copy(c, c + 4, ostream_iterator<int> (cout, " "));
cout << endl;
```

数值 - 随机数

- 随机数 (Random Numbers)
  - 为使每次的随机数序列都不一样，需动态改变种子的值，即每次传给 srand() 的参数不同

```
// in <cstdlib>
// Return a random integer between 0 and RAND_MAX inclusive.
extern int rand (void);
// Seed the random number generator with the given number.
extern void srand (unsigned int seed);
```

```
    srand((unsigned) time(0));

    int rd = 0;
    for (int i = 0; i < 10; ++i) {
        // rd = rand() % 100;
        rd = int((double(rand()) / RAND_MAX) * 100);
        cout << rd << ' ';
    }
    cout << endl;
```

数值 - Bjarne's Advices

- Bjarne's Advices
  - 数值问题常常很微妙，如果对数值问题的数学方面不是完全有把握，请去找专家或做试验
  - 用切割描述在数组的一部分上操作，而不是用循环
  - 在写循环从一个序列出发计算某个值前，先考虑accumulate()、 inner_product()、 partial_sum()、adjacent_difference()
  - 注意使你的随机数充分随机