CS601: Software Development for Scientific Computing

Autumn 2021

Week5:

- Intermediate C++ (template programming and STL), Structured Grids (Elliptic PDEs)

Last Week..

- Tools
  - GNU `make`, `git`
- Intermediate C++
  - Object Orientation: inheritance, polymorphism, abstract base classes, (about const, references)
  - function templates

Function Templates - Recap

<pre>double scprod(int len,
          double* vec1,
          double* vec2)
{
    double result;
    //compute result
    //return result
}</pre>
	<pre>int scprod(int len,
	          int* vec1,
	          int* vec2)
	{
	    int result;
	    //compute result
	    //return result
	}</pre>

How can you avoid multiple implementations of the same functionality but with different types?

Function Templates - Recap

```
template<typename T>
double scprod(int len,
         T* vec1,
         T* vec2)
{
    T result;
    //compute result
    //return result
}
```

Add this template definition in .h file! why .h and not .cpp?

Called template parameter. Can choose any name other than T. the keyword ‘typename’ can be replaced with ‘class’

```
int main() {
//define vec1-vec4
scprod<double>(10,vec1, vec2); //explicit instantiation
scprod<int>(100,vec3,vec4); //explicit instantiation
scprod(100, vec3,vec4); //implicit instantiation
```

Class Templates

- Like function templates but for templating classes
  - Refer to templates_class in week5_codesamples

Standard Template Library (STL)

- Large set of frequently used data structures and algorithms
  - Defined as *parametrized* data types and functions
  - Types to represent complex numbers and strings, algorithms to sort, get random numbers etc.
- Convenient and bug free to use these libraries
- E.g. `vector, map, queue, pair, sort` etc.
- Use your own type only for efficiency considerations - *only if you are sure!*

STL - Motivation

Real-world view
source:wikipedia

Coconut meat, raw

Nutritional value per 100 g (3.5 oz)		
Energy	354 kcal (1,480 kJ)	
Carbohydrates	15.23 g	
Sugars	6.23 g	
Dietary fiber	9.0 g	
Fat	33.49 g	
Saturated	29.698 g	
Monounsaturated	1.425 g	
Polyunsaturated	0.366 g	
Protein	3.33 g	
Tryptophan	0.039 g	
Threonine	0.121 g	
Isoleucine	0.131 g	
Leucine	0.247 g	
Lysine	0.147 g	
Methionine	0.062 g	
Cystine	0.066 g	
Phenylalanine	0.169 g	
Tyrosine	0.103 g	
Valine	0.202 g	
Arginine	0.546 g	
Histidine	0.077 g	
Alanine	0.170 g	
Aspartic acid	0.325 g	
Glutamic acid	0.761 g	
Glycine	0.158 g	
Proline	0.138 g	
Serine	0.172 g	
Vitamins	**Quantity**	**%DV**†

Consider the nutrients (constituents) present in edible part of coconut.
How would you capture the Real-world view in a Program?

```
vector<pair<string, float> > constituents;
```

Container

- Holder of a collection of objects
- Is an object itself
- Different types:
  - sequence container
  - associative container (ordered/unordered)
  - container adapter

Sequence Container

- Provide fast sequential access to elements
- Factors to consider:
  - Cost to add/delete an element
  - Cost to perform non-sequential access to elements

container name	**comments**
`vector`	Flexible array, fast random access
`string`	Like `vector`. Meant for sequence of characters
`list/slist`	doubly/singly linked list. Sequential access to elements (bidirectional/unidirectional).
`deque`	Double-ended queue. Fast random access, Fast append
`array`	Intended as replacement for ‘C’-style arrays. Fixed-sized.

Container Adapter

- Provide an interface to sequence containers
  - `stack, queue, priority_queue`

Associative Container

- Implement sorted data structures for efficient searching (O(log n)) complexity.
  - Set, map, multiset, multimap

container name	**comments**
`set`	Collection of unique sorted keys. Implemented as class template
`map`	Collection of key-value pairs sorted by unique keys. Implemented as class template

Unordered Associative Container

- Implement hashed data structures for efficient searching (O(1) best-case, O(n) worst-case complexity).
  - `unordered_set`, `unordered_map`, `unordered_multiset`, `unordered_multimap`

Vectors

- An array that expands and shrinks automatically
  - Parametrized data structure
- E.g.
  - `std::vector<int> integers;`

```
//empty array that can hold integer numbers
```

  - `std::vector<Fruit> fruits(10);`

```
//array of 10 elements of type Fruit. The 10 objects are
initialized by //invoking default constructor
```

  - Recall:

Type for a pair of any types (`type1, type2`)

```
class Coconut {
vector<pair<string, float> > constituents;
...
```

Vectors – adding elements

Object creation and initialization

```
#include<vector> //in Fruit.h

int main() {
        Coconut* c;
        c=Coconut(“Coconut”,1.2)
        //..
}

Coconut::Coconut(string name, float weight) : Fruit(name, weight) {
        constituents.push_back(make_pair(“sugars”,6.23));
        constituents.push_back(make_pair(“fiber”,9));
        //...
}
```

Vectors – Object Layout

Object layout in memory

Fruit part of the object:

```
commonName = "Coconut"
Weight = 1.2
energyPerUnitWeight = 3.6
vptr = ...
```

Coconut part of the object:

```
constituents = {
<sugars,6.23>,
<fiber, 9>,
<saturated_fat, 29.69>,
<water, 47g>,
}
```

Vectors – operations

declaration: `vector<pair<string, float> > constituents;`

- **Reading elements:**

```
constituents.push_back(make_pair("sugars",6.23))
pair<string, float> tmpVal = constituents[0];
```

- **Removing elements:**

```
constituents.push_back(make_pair("fiber",9))
constituents.pop_back();
```

- **Finding number of elements:**

```
cout<<constituents.size()<<endl;
```

Vectors – operations

`declaration: vector<pair<string, float> > constituents;`

Element-wise inspection (iterating over vector elements):

```
vector<pair<string, float>::iterator it;
for(it=constituents.begin(); it!=constituents.end(); it++) {
       pair<string, float> elem = *it;
       cout<<elem.first<<“,”<<elem.second<<endl;
       //can also use cout<<it->first<<“,”<<it->second<<endl;
}
```

Reference: http://www.cplusplus.com/reference/vector/vector/

sort

- Sort fruits by their weight / energy / name

```
#include<algorithm>
bool comp(Fruit* obj1, Fruit* obj2) {
        if(obj1->GetWeight() < obj2->GetWeight())
                return true;
        return false;
}
```

```
int main() {
    Apple* a1=new Apple("Apple",0.24);
    Orange* o=new Orange("Orange",0.15);
    Mango*     m=new Mango("Mango",0.35);
    Apple* a2=new Apple("Apple",0.2);
    vector<Fruit*> fruits;
    fruits.push_back(a1);
    fruits.push_back(o);
    fruits.push_back(m);
    fruits.push_back(a2);
    sort(fruits.begin(),fruits.end(),comp);
}
```

Exceptions

- Preferred way to handle logic and runtime errors
  - Unhandled exceptions stop program execution. Handle exceptions and recover from errors.
  - Clean separation between error detection and handling.
- Where to use? often in `public` functions
  - no control over arguments passed
- Are there performance penalties?
  - Mostly not. ‘exceptions’: memory-constrained devices, real-time performance requirements

Exceptions

- E.g.

```
Fruit::Fruit(string name, float wt) {
        if(wt < 0)
                throw std::invalid_argument("Invalid weight");
        }
        ...
}

int main() {
        try {
                Apple* a = new Apple("Apple_gala",-0.4);
        }catch(const std::invalid_argument& ia) {
                cerr<<ia.what()<<endl;
        }
}
```

keywords

reference: http://www.cplusplus.com/doc/tutorial/exceptions/

Post-class Exercise – STL and Exceptions

Reattempt the same quiz on STL and Exceptions

When do we need to return reference to an object? Why?

Returning References- Example1

- How can we assign one object to another?

```
Apple a1("Apple", 1.2); //constructor Apple::Apple(string, float)
                         //is invoked
Apple a2; //constructor Apple::Apple() is invoked.
a2  = a1 //object a1 is assigned to a2;assignment operator is invoked
```

```
Apple& Apple::operator=(const Apple& rhs)
```

Called Copy Assignment Operator

```
Apple& Apple::operator=(const Apple& rhs) {
commonName = rhs.commonName;
weight = rhs.weight;
energyPerUnitWeight = rhs.energyPerUnitWeight;
constituents = rhs.constituents;
return *this;
}
```

`this`

- Implicit variable defined by the compiler for every class
  - `E.g. MyVec *this;`
- All member functions have `this` as an implicit first argument
  - E.g.

```
int MyVec::GetVecLen() const;
```

would actually be:

```
int MyVec::GetVecLen(MyVec* this) const;
```

Returning References – Example2

```
#ifndef MYVEC_H
#define MYVEC_H
class MyVec{
        //private attributes
        double* data;
        int vecLen;
public:
        MyVec(int len); //constructor decl.
        MyVec(const MyVec& rhs); //copy cons
tructor
        int GetVecLen() const; //member func
tion
        double& operator[](int index) const;
        ~MyVec(); //destructor decl.


};
```

```
}

MyVec::MyVec(const MyVec& rhs) {
        vecLen=rhs.GetVecLen();
        data=new double[vecLen];
        for(int i=0;i<vecLen;i++) {
                data[i] = rhs[i];
        }
}

//defining GetVecLen member function
int MyVec::GetVecLen() const {
        return vecLen;
}

double& MyVec::operator[](int index) const {
        return data[index];
}
```

```
MyVec v1;
v1[0]=100;
```

L-values and R-values

- **L-values**: addresses which can be loaded from or stored into
- **R-values**: data often loaded from address
  - Expressions produce R-values
- Assignment statements: `L-value := R-value;`

```
i := 5;        //RHS specifies data that is computed/read.
i := i + 1;    //LHS specifies address where data is stored.
```

```
a := a;
```

a refers to memory location named a (L-value). We are storing into that memory location

a refers to data stored in the memory location named a. We are loading from that memory location to produce R-value

Overloading +=

- `MyVec v1;`

  `v1+=3;`
- `MyVec& MyVec::operator+=(double)`

Overloading +=

- `MyVec v1;`

  `v1+=3;`
  - `MyVec& MyVec::operator+=(double)`
- `MyVec v2;`

  `v2+=v1;`
  - `MyVec& MyVec::operator+=(const MyVec& rhs)`
  - What if you make the return value above `const`?

    Disallow: `(v2+=v1)+=3;`

Overloading +

- `v1=v1+3;` ***Single-argument constructors:** allow implicit conversion from a particular type to initialize an object.*
  - `const MyVec MyVec::operator+(double val)`
- `v3=v1+v2;`

  1. `const MyVec MyVec::operator+(const MyVec& vec2) const;`

OR

  2. `friend const MyVec operator+(const MyVec& lhs, const MyVec& rhs);`

v1=3+v1 is compiler error! Why?

Operator Overloading - Guidelines

- If a binary operator accepts operands of different types and is commutative, both orders should be overloaded
- Consistency:
  - If a class has ==, it should also have !=
  - += and + should result in identical values
  - define your copy assignment operator if you have defined a copy constructor

Exercise

- What member functions does class `MyVec` should define to support:

```
MyVec v2;
v2=-v1; //v1 is of type MyVec
```

- Bonus: How to define pre-increment (`++obj`) and post-increment (`obj++`) operations?

PDEs - Recap

- consider a function $u = u(x, t)$ satisfying the second-order PDE:

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial t} + C\frac{\partial^2 u}{\partial t^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial t} + Fu = G\ ,$$

Where A-G are given functions. This is a PDE of type:

- Parabolic: if $B^2 - 4AC = 0$
- Elliptic: if $B^2 - 4AC < 0$
- Hyperbolic: if $B^2 - 4AC > 0$

Important PDEs - Recap

Laplace operator $(\mathbf{L})$: of a two-times continuously differentiable scalar-valued function $u: \mathbb{R}^n \to \mathbb{R}$

$$\Delta u \quad = \sum_{k=1}^{n} \partial_{kk} u$$

- **Poisson problem**: $-\Delta u = f$ (elliptic, independent of time.)
- **Heat equation**: $\partial_t u - \Delta u = f$ (parabolic. Here, $\partial_t u = \frac{\partial u}{\partial t}$ = partial derivative w.r.t. time)
- **Wave equation**: ${\partial_t}^2 u - \Delta u = f$ (Hyperbolic. Here, ${\partial_t}^2 u = \frac{\partial^2 u}{\partial t \partial t}$ = second-order partial derivative w.r.t. time)

Definitions - Recap

- Consider a region of interest R in, say, xy plane. The following is a *boundary-value problem*:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$,where

f is a given function in R and

$u = g$,where

the function g tells the value of function u at boundary of R

- if $f = 0$ everywhere, then Eqn. (1) is Laplace's Equation
- if $f \neq 0$ somewhere in R, then Eqn. (1) is Poisson's Equation

Exercise

- Consider the *boundary-value* problem:

$u_{xx} + uyy = 0$ in the square $0 < x < 1, 0 < y < 1$

$u = x^2y$ on the boundary.

Is this Laplace equation or Poisson equation?

Elliptic Equation – Numerical Solution

- Approximate the derivatives of $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$ using central differences
- Choose step sizes δx and δy for x and y axis resp.
  - Both and x and y are independent variables here.
  - Choose δx= δy = h
- Write difference equation for approximating the PDE above