**CH1-8计算器函数**

**class Token\_stream {**

**bool full; // *is there a Token in the buffer?***

**Token buffer; // *where we keep a Token put back using putback()***

**public: Token get(); // *get a Token***

**void putback(Token); // *put a Token back into the Token\_stream***

**Token\_stream(); // *constructor: make a Token\_stream***

**void ignore(char c); }; // *discard tokens up to and including***

**void Token\_stream::putback(Token t)**

**{ if (full) error("putback() into a full buffer");**

**buffer=t; full=true; }**

**Token Token\_stream::get() {**  *// read a Token from the Token\_stream*

**if (full) { full=false; return buffer; }** // *we already have a Token*

**char ch;**

**cin >> ch; // *note that >> skips whitespace (space, newline, tab, etc.)***

**switch (ch) {**

**case '(': case ')': case ';': case 'q': case '+': case '-': case '\*': case '/':**

**return Token(ch);** // *let each character represent itself*

**case '.':**

**case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9': {**

**cin.putback(ch);** // *put digit back into the input stream*

**double val;**

**cin >> val;** // *read a floating-point number*

**return Token(number,val);** // no *‘8’ represent “a number”*

**}**

**default: error("Bad token");**

**}}**

**void Token\_stream::ignore(char c)**  
{ if (full && c == buffer.kind) {full = false; return; }  
 full = false;  
 char ch;  
 while (cin >> ch) if (ch == c) return;}

**double expression() // *read and evaluate: 1 1+2.5 1+2+3.14 etc.***

**{ double left = term(); // *get the Term***

**while (true) {**

**Token t = get\_token(); // *get the next token…***

**switch (t.kind) { // *… and do the right thing with it***

**case '+': left += term(); break;**

**case '-': left -= term(); break;**

**default: ts.putback(t); // *put the unused token back***

**return left; }**

**}}**

**double term() //** *exactly like expression(), but for \* and /*

**{** **double left = primary();** // *get the Primary*

**while (true) {**

**Token t = get\_token();** // *get the next Token*

**switch (t.kind) {**

**case '\*': left \*= primary();**

**break;**

**case '/‘: double d = primary();**

**if (d==0) error("divide by zero");**

**left /= d;**

**break;**

**default:**

**return left;** // *return the value*

**}}}**

**double primary() // *Number or ‘(‘ Expression ‘)’***

**{ Token t = get\_token();**

**switch (t.kind) {**

**case '(': // *handle ‘(’expression ‘)’***

**double d = expression();**

**t = get\_token();**

**if (t.kind != ')') error("')' expected" );**

**return d;**

**case '8': // *we use ‘8’ to represent the “kind” of a number***

**return t.value; // *return the number’s value***

**case ‘-’: //处理负数**

**return 0- primary();**

**default:**

**ts.putback(t); // *put unused token back into input stream***

**error("primary expected");}}**

**CH10-11 iostream**

iostream 将所有的错误归结为四类流状态

good() // *the operation succeeded*

eof() // *we hit the end of input (“end of file”)*

fail() // *something unexpected可恢复（数据格式错误）*

bad()// *something unexpected and serious happened不可恢复*

**void skip\_to\_int() //跳过无效字符**

**{** **if (cin.fail()) { //** *we found something that wasn’t an integer*

**cin.clear(); //** *we’d like to look at the characters*

**char ch;**

**while (cin>>ch) { //** *throw away non-digits*

**if (isdigit(ch)) {**

**cin.unget(); //** *put the digit back,*

**//** *so that we can read the number*

**return;}}}**

**error("no input"); //** *eof or bad: give up*

**}**

**int get\_int() //获取任意整数**

**{** **int n = 0;**

**while (true) {**

**if (cin >> n) return n;**

**cout << "Sorry, that was not a number; please try again\n";**

**skip\_to\_int();}}**

**int get\_int(int low, int high) //读取给定范围内的整数**

**{ cout << "Please enter an integer in the range "**

**<< low << " to " << high << " (inclusive):\n";**

**while (true) {**

**int n = get\_int();**

**if (low<=n && n<=high) return n;**

**cout << "Sorry, "**

**<< n << " is not in the [" << low << ':' << high**

**<< "] range; please try again\n";}}**

默认情况下，ifstream 打开文件为读模式

默认情况下，ofstream 打开文件为写模式

替代选项：

ios\_base::app // *append (i.e., add to the end of the file)*

ios\_base::ate // *“at end” (open and seek to end)*

ios\_base::binary // *binary mode – beware of system specific behavior*

ios\_base::in // *for reading*

ios\_base::out // *for writing*

ios\_base::trunc // *truncate file to 0-length*

在文件名之后，指定文件打开模式是可选的

ofstream of1(name1); // *defaults to ios\_base::out*

ifstream if1(name2); // *defaults to ios\_base::in*

ofstream ofs(name, ios\_base::app); // *append rather than overwrite*

fstream fs("myfile", ios\_base::in|ios\_base::out); // *both in and out*

**CH12-16 GUI函数**

**class Shape {** // deals with color and style, and holds a sequence of lines

**public:**

**void draw() const; //** *deal with color and call draw\_lines()*

**virtual void move(int dx, int dy); //** *move the shape +=dx and +=dy*

**void set\_color(Color col); //** *color access*

**int color() const;** **//** *… style and fill\_color access functions …*

**Point point(int i) const; //** *(read-only)* access to points

**int number\_of\_points() const;**

**protected:**

**Shape(); //** *protected to make class Shape abstract*

**void add(Point p); //** *add p to points*

**virtual void draw\_lines() const; //** *draw the appropriate lines*

**private:**

**vector<Point> points; //** *not used by all shapes*

**Color lcolor; //** *line color*

**Line\_style ls; //** *line style*

**Color fcolor; //** *fill color*

***Shape(const Shape&); //*** *can’t copy construct as it’s private*

***Shape& operator=(const Shape&); //*** *don’t copy assign as it’s private*

**};**

**struct Lines\_window : Window //** *Lines\_window inherits from Window*

**{**

**Lines\_window(Point xy, int w, int h, const string& title); //** *declare constructor*

**Open\_polyline lines;**

**private:**

**Button next\_button; //** *declare some buttons – type Button*

**Button quit\_button;**

**In\_box next\_x; //** *declare some i/o boxes*

**In\_box next\_y;**

**Out\_box xy\_out;**

**void next(); //** *what to do when next\_button is pushed*

**void quit(); //** *what to do when quit\_botton is pushed*

**staticvoid cb\_next(Address, Address window); //** *callback for next\_button*

**static void cb\_quit(Address, Address window); //** *callback for quit\_button***};**

**Lines\_window::Lines\_window(Point xy, int w, int h, const string& title)**

**:Window(xy,w,h,title), //** *construct/initialize the parts of the window*

**next\_button(Point(x\_max()-150,0), 70, 20, "Next point", cb\_next),**

**quit\_button(Point(x\_max()-70,0), 70, 20, "Quit", cb\_quit), //** *quit button*

**next\_x(Point(x\_max()-310,0), 50, 20, "next x:"), //** *io boxes*

**next\_y(Point(x\_max()-210,0), 50, 20, "next y:"),**

**xy\_out(Point(100,0), 100, 20, "current (x,y):")**

**{**

**attach(next\_button); //** *attach the parts to the window*

**attach(quit\_button);**

**attach(next\_x);**

**attach(next\_y);**

**attach(xy\_out);**

**attach(lines); //** *attach the open\_polylines to the window* **}**

**void Lines\_window::next() {**

**int x = next\_x.get\_int();**

**int y = next\_y.get\_int();**

**lines.add(Point(x,y));**

**//** *update current position readout:*

**stringstream ss;**

**ss << '(' << x << ',' << y << ')';**

**xy\_out.put(ss.str());**

**redraw(); //** *now redraw the screen***}**

**struct Button : Widget {**

**Button(Point xy, int w, int h, const string& s, Callback cb)**

**:Widget(xy,w,h,s,cb) { }};**

**void Lines\_window::cb\_next(Address, Address pw) { //回调**

**reference\_to<Lines\_window>(pw).next() //** *now call our function***}**

**struct In\_box : Widget {**

**In\_box(Point xy, int w, int h, const string& s)**

**:Widget(xy,w,h,s,0) { }**

**int get\_int();**

**string get\_string();};**

**int In\_box::get\_int(){**

**//** *get a reference to the FLTK FL\_Input widget:*

**Fl\_Input& pi = reference\_to<Fl\_Input>(pw);**

**//** *use it:*

**return atoi(pi.value()); //** *get the value and convert*

**//** *it from characters (****a****lpha)* ***to******i****nt* **}**

**CH19-Vector终版实现**

**template<class T> class vector { // double换成T**

**int sz; //** *the size*

**double\* elem; //** *a pointer to the elements*

**int space; //** *size+free\_space*

**public:**

**vector() : sz(0), elem(0), space(0) { } //***default**constructor*

**explicit vector(int s) :sz(s), elem(new double[s]) , space(s) { } //** *constructor*

**vector(const vector&); //** *copy constructor*

**vector& operator=(const vector&); //***copy assignment*

**~vector() { delete[ ] elem; }**  **//** *destructor*

**double& operator[ ](int n) { return elem[n]; } //** *access: return reference*

**int size() const { return sz; } //** *current size*

**int capacity() const { return space; } //***current space*

**void resize(int newsize); //** *grow (or shrink)*

**void push\_back(double d); //** *add element*

**void reserve(int newalloc); //** *get more space*

**T& at(int n); //** *checked access*

**const T& at(int n) const; //** *checked access*

**T& operator[ ](int n); //** *unchecked access*

**const T& operator[ ](int n) const; //***unchecked access*

**};**

**void vector::reserve(int newalloc)** {

// 为原有向量,指定一个新大小的保留空间

// 新的大小，只能比原大小更大，才会有效

**if (newalloc<=space) return; //** *never decrease allocation*

**double\* p = new double[newalloc]; //** *allocate new space*

**for (int i=0; i<sz; ++i) p[i]=elem[i]; //** *copy old elements*

**delete[ ] elem; //** *deallocate old space*

**elem = p;**

**space = newalloc;}**

**void vector::resize(int newsize) {**

**//** *make the vector have* ***newsize*** *elements*

// *initialize each new element with the default value 0.0*

**reserve(newsize); //** *make sure we have sufficient space*

**for(int i = sz; i<newsize; ++i) elem[i] = 0; //** *initialize new elements*

**sz = newsize; }**

**void vector::push\_back(double d){ //** *increase**vector size**by one*

// *initialize the new element with* ***d***

**if (sz==0) //** *no space: grab some*

**reserve(8);**

**else if (sz==space) //** *no more free space:**get more space*

**reserve(2\*space);**

**elem[sz] = d; //** *add* ***d*** *at end*

**++sz; //** *increase the size (****sz*** *is the number of elements)*

**}**

**vector& vector::operator=(const vector& a)**

**{**

**if (this==&a) return \*this;** // *self-assignment, no work needed*

**if (a.sz<=space) {** // *enough space, no need for new allocation*

**for (int i = 0; i<a.sz; ++i) elem[i] = a.elem[i];**//*copy elements*

**sz = a.sz;**

**return \*this;}**

**double\* p = new double[a.sz];**  //没有条件只好 *copy and swap*

**for (int i = 0; i<a.sz; ++i) p[i] = a.elem[i];**

**delete[ ] elem;**

**sz = a.sz;**

**space = a.sz;**

**elem = p;**

**return \*this; }**

**template<class T>**

**T& vector<T>::at (int n){**

**if (n<0 || sz<=n) throw out\_of\_range();**

**return elem[n]; }**

**template<class T>**

**T& vector<T>::operator[ ](int n){**

**return elem[n];** **}**

**CH20/21-容器、迭代器和算法**

**template<class T, class A=allocator<T> >**

**class vector {** **//…**

**Public:**

**typedef unsigned long size\_type;**

**typedef T value\_type;**

**typedef T\* iterator; //** *the type of an iterator is implementation defined* // *a vector iterator could be a pointer to an element*

**typedef const T\* const\_iterator;**

**iterator begin(); //** *points to first element*

**const\_iterator begin() const;**

**iterator end(); //** *points one beyond the last element*

**const\_iterator end() const;**

**iterator erase(iterator p); //** *remove element pointed to by* ***p***

**iterator insert(iterator p, const T& v); //** *insert a new element* ***v*** *before* ***p*};**

**template<class T, class A>**

**vector<T,A>::iterator vector<T,A>::erase (iterator p)**

**{**  **if (p==end()) return p;**

**for (iterator pos = p+1; pos!=end(); ++pos)**

**\*(pos-1) = \*pos;** //拷贝元素到其左边的一个位置

**alloc.destroy(&\*(end()-1);** *//*连带拷贝地销毁最后一个单元

**return p;};**

**template<class T, class A>**

**vector<T,A>::iterator vector<T,A>::insert(iterator p, const T& va)**

**{**  **int index = p-begin();**

**if (size()==capacity()) reserve(size()==0? 8:2\*size;** //确保有足够空间

//首先拷贝最后一个元素到未初始化空间

**alloc.construct(elem+sz, \*back());**

**++sz;**

**iterator pp = begin() + index; //** *the place to put val*

**for (iterator pos = end()-1; pos!=pp; --pos)**

**\*pos = \*(pos-1);** //拷贝元素到其右边的一个位置

**\*(begin() + index) = val;** *//”insert” val*

**return pp;};**

谓词是一个函数或函数对象，它获得参数后返回一**bool** 值

**template<class In, class Pred>**

**In find\_if(In first, In last, Pred pred) {**// p**red**是基于类Pred构造的对象

**while (first!=last && !pred(\*first)) ++first;** //此处,谓词限定带一个参数的调用

**return first**; //传入的函数参数接口，必须与此限定一致**}**

**template<class In, class Out, class Pred>**

**Out copy\_if(In first, In last, Out res, Pred p) {// *copy elements that fulfill the predicate***

**while (first!=last) {**

**if (p(\*first)) \*res++ = \*first;**

**++first;}**

**return res;}**

**template<class T>**

**struct Less\_than {**

**T val; // *value to compare with***

**Less\_than(int x) :val(x) { } //*构造***

**bool operator()(const T& x) const { return x < val; } //*操作符()***

**};**

**template<class In, class T, class BinOp>**

**T accumulate(In first, In last, T init, BinOp op) {**

**while (first!=last) {**

**init = op(init, \*first); //** *means “init op \*first”*

**++first;}**

**return init;**

**}**

**map<string,int> fruits** 事实上是一个有序平衡二叉树，而

**set<string> fruits**实际上是一个有序平衡二叉树，没有值类型，不支持[]索引取值，新值插入位置由容器决定，不支持push\_back()操作

**template<class Key, class Value> class map {**

**typedef pair<Key,Value> value\_type; //** *a map in (Key,Value) pairs*

**typedef ??? iterator; //** *probably a pointer to a tree node*

**typedef ??? const\_iterator;**

**iterator begin(); //** *points to first element*

**iterator end(); //** *points to one beyond the last element*

**Value& operator[ ](const Key&);**

**iterator find(const Key& k); //** *is there an entry for* ***k?***

**void erase(iterator p); //** *remove element pointed to by* ***p***

**pair<iterator, bool> insert(const value\_type&); //** *insert a new pair before* ***p*** **};**