```
#ifndef _ANDFILTER_H
#define _ANDFILTER_H

/**
 * This filter class works like a Decorator
 * client code calls f->ok(..) for a Filter f,
 *
 * @author Owen Astrachan
 */

#include "filter.h"

class AndFilter : public Filter
{
  public:
    AndFilter(Filter * a, Filter * b);
    virtual ~AndFilter() {};

    virtual bool operator()(const DirEntry& entry) const;

  private:
    Filter * myFirst;
    Filter * mySecond;
};

#endif
```

```
#ifndef _FILTER_H
#define _FILTER_H

/**
 * This filter class works like a Decorator
 * client code calls f->ok(..) for a Filter f,
 *
 * @author Owen Astrachan
 */

class DirEntry;

#include "directory.h"
#include <iostream>
using namespace std;

class Filter
{
  public:
    Filter(Filter * filter) : myFilter(filter){ }

    virtual ~Filter() {};

    virtual bool operator() (const DirEntry& entry) const
    {
        return myFilter->operator()(entry);
    }

  protected:
    Filter() : myFilter(this) { }

  private:
    Filter * myFilter;

};

class TrueFilter : public Filter
{
  public:
    virtual bool operator() (const DirEntry& entry) const
    {
        return true;
    }
};

class FalseFilter : public Filter
{
  public:
    virtual bool operator() (const DirEntry& entry) const
    {
        return false;
    }
};

class NotFilter : public Filter
{
  public:
    NotFilter(Filter * f) : Filter(this), myFilter(f) {
    }
    virtual bool operator()(const DirEntry& entry) const
    {
        return ! myFilter->operator()(entry);
    }
  private:
    Filter * myFilter;

};

#include "andfilter.h"
```



```
#include "regexpfilter.h"
#include "orfilter.h"

#endif
```

```
#ifndef _FORMAT_H
#define _FORMAT_H

class DirEntry;

class Format
{
  public:
    virtual string format(const DirEntry& entry);
};

class LongFormat : public Format
{
    virtual string format(const DirEntry& entry);
};

#endif
```

```
/*
 * Copyright (c) 1987, 1993, 1994, 1996
 *  The Regents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by the University of
 *  California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#ifndef __GETOPT_H__
#define __GETOPT_H__

#ifdef __cplusplus
extern "C" {
#endif

extern int   opterr;      /* if error message should be printed */
extern int   optind;      /* index into parent argv vector */
extern int   optopt;      /* character checked for validity */
extern int   optreset;    /* reset getopt */
extern char *optarg;      /* argument associated with option */

int getopt (int, char * const *, const char *);

#ifdef __cplusplus
}
#endif

#endif /* __GETOPT_H__ */

#ifndef __UNISTD_GETOPT__
#ifndef __GETOPT_LONG_H__
#define __GETOPT_LONG_H__

#ifdef __cplusplus
extern "C" {
#endif

struct option {
        const char *name;
        int  has_arg;
        int *flag;
        int val;
};
```



```
int getopt_long (int, char *const *, const char *, const struct option *, int *)
;
#ifndef HAVE_DECL_GETOPT
#define HAVE_DECL_GETOPT 1
#endif

#define no_argument             0
#define required_argument       1
#define optional_argument       2

#ifdef __cplusplus
}
#endif

#endif /* __GETOPT_LONG_H__ */
#endif /* __UNISTD_GETOPT__ */
```

```
#ifndef _ITERADAPTER_H
#define _ITERADAPTER_H

template <class Entry, class Iterator>
class iteradapter
{
  public:

    // define typedefs expected by STL input iterators

    typedef input_iterator_tag iterator_category;
    typedef Entry value_type;
    typedef ptrdiff_t difference_type;
    typedef const Entry * pointer;
    typedef const Entry&  reference;


    iteradapter()
        : myIterator(0),
          myPastEnd(true)
    //  post: nothing to read, all done (one past end iterator)
    {

    }

    iteradapter(Iterator& ds)
      : myIterator(&ds),
        myPastEnd(false)
    // post: bound to ds, iterate over all entries
    {
        ds.Init();
        getEntry();
    }

    static iteradapter<Entry,Iterator> begin(Iterator& ds)
    {
        return iteradapter(ds);
    }

    static iteradapter<Entry,Iterator> end()
    {
        return iteradapter();
    }

    reference operator*() const
    // pre: ! myPastEnd
    // post: return current entry in stream
    {
        return myEntry;
    }

    pointer operator->() const
    // pre: ! myPastEnd
    // post: return pointer to current entry
    {
        return &myEntry;
    }

    iteradapter operator++()
    // post: entry updated (this is pre-increment)
    {
        getEntry();
        return *this;
    }

    iteradapter operator++(int)
    // post: entry updated (this is post-increment)
    {
        iteradapter tmp = *this;
```



```
        getEntry();
        return tmp;
    }

    bool operator==(const iteradapter& rhs) const
    // post: return true iff *this and *rhs both past end OR
    //       if they share the same stream and have the same "end-ness"
    {
        return (myIterator == rhs.myIterator && myPastEnd == rhs.myPastEnd) ||
               (myPastEnd == true && rhs.myPastEnd == true);
    }

    bool operator!=(const iteradapter& rhs) const
    {
        return !( *this == rhs);
    }


  private:

    void getEntry()
    {
        if (myIterator->HasMore())
        {
            myEntry = myIterator->Current();
            myIterator->Next();
        }
        else
        {
            myPastEnd = true;
        }
    }
    Iterator * myIterator;   // wrapper for this iterator
    Entry      myEntry;      // current entry
    bool       myPastEnd;    // true iff one past end
};

#endif
```

```
#ifndef _OOLSPROCESSOR_H
#define _OOLSPROCESSOR_H

class Filter;
class Sorter;
class Format;
class Printer;

class OOLSprocessor
{
  public:
    OOLSprocessor();
    virtual void process(const string& dirname,
                         Sorter * sorter,
                         Printer * printer,
                         Format * format,
                         Filter * filter);

  protected:

    Sorter * mySorter;
    Filter * myFilter;
    Format * myFormat;
    Printer * myPrinter;
};

#endif
```

```
#ifndef _ORFILTER_H
#define _ORFILTER_H

/**
 * This filter class works like a Decorator
 * client code calls f->ok(..) for a Filter f,
 *
 * @author Owen Astrachan
 */

#include "filter.h"

class OrFilter : public Filter
{
  public:
    OrFilter(Filter * a, Filter * b);
    virtual ~OrFilter() {};

    virtual bool operator()(const DirEntry& entry) const;

  private:
    Filter * myFirst;
    Filter * mySecond;
};

#endif
```

```
#ifndef _PRINTER_H
#define _PRINTER_H

#include <string>
#include <iostream>
#include <vector>
using namespace std;


class Printer
{
  public:
    Printer();
    virtual ~Printer(){}
    virtual void setCols(int cols) { }
    virtual void print(ostream& out, vector<string>& list) const;
};


class ColumnPrinter : public Printer
{
  public:
    ColumnPrinter();
    virtual void print(ostream& out, vector<string>& list) const;
    virtual void setCols(int cols);

  private:

    int myCols;
};

#endif
```

```
#ifndef _REGEXPFILTER_H
#define _REGEXPFILTER_H

/**
 *
 * @author Owen Astrachan
 */

#include <string>
using namespace std;

#include "filter.h"

class RegExpFilter : public Filter
{
  public:
    RegExpFilter(const string& s);
    virtual ~RegExpFilter() {};

    virtual bool operator()(const DirEntry& entry) const;

  private:
    string myString;

};

#endif
```

```
#ifndef _SIZEFILTER_H
#define _SIZEFILTER_H

/**
 * This filter class works like a Decorator
 * client code calls f->ok(..) for a Filter f,
 *
 * @author Owen Astrachan
 */

#include "filter.h"

class SizeFilter : public Filter
{
  public:
    SizeFilter(int size);
    virtual ~SizeFilter() {};

    virtual bool ok(const DirEntry& entry) const;

  private:

    int mySize;
};

#endif
```

```
#ifndef _SORTER_H
#define _SORTER_H

class DirEntry;

class Sorter
{
  public:
    Sorter() : mySorter(this){
    }
    Sorter(Sorter * s) : mySorter(s){
    }
    virtual bool operator() (const DirEntry& lhs,
                             const DirEntry& rhs) const
    {
            return (*mySorter)(lhs,rhs);
    }

  private:

    Sorter * mySorter;
};

class AlphSorter : public Sorter
{
  public:
    virtual bool operator() (const DirEntry& lhs, const DirEntry& rhs) const{
        return lhs.Name() < rhs.Name();
    }
};

class SizeSorter : public Sorter
{
  public:
    bool operator() (const DirEntry& lhs, const DirEntry& rhs) const{
        return lhs.Size() < rhs.Size();
    }
};

class TimeSorter : public Sorter
{
  public:
    bool operator() (const DirEntry& lhs, const DirEntry& rhs) const{
        if (lhs.GetDate() == rhs.GetDate()){
            return lhs.GetTime() < rhs.GetTime();
        }
        return lhs.GetDate() < rhs.GetDate();
    }
};

class ReverseSorter : public Sorter
{
  public:
    ReverseSorter(Sorter * s)
        : mySorter(s)
    { }

    bool operator() (const DirEntry& lhs, const DirEntry& rhs) const{
        return ! (*mySorter)(lhs,rhs);
    }

  private:
    Sorter * mySorter;
};

#endif
```

```
#include "andfilter.h"
#include "directory.h"

AndFilter::AndFilter(Filter * a, Filter * b)
    : myFirst(a),
      mySecond(b)
{

}

bool AndFilter::operator()(const DirEntry& entry) const
{
    return (*myFirst)(entry) && (*mySecond)(entry);
}
```

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
#include "directory.h"
#include "sizefilter.h"
#include "regexpfilter.h"
#include "andfilter.h"


bool operator < (const DirEntry& lhs, const DirEntry& rhs)
{
    return lhs.Name() < rhs.Name();
}


bool sizecomp (const DirEntry& lhs, const DirEntry& rhs)
{
    return lhs.Size() < rhs.Size();
}



void printAll (DirStream& dir, Filter* filter)
{
    vector<DirEntry> files;
    for (dir.Init(); dir.HasMore(); dir.Next())
    {
        DirEntry entry = dir.Current();
//      if (filter->ok(entry))
        if ((*filter)(entry))
        {
            files.push_back(entry);
        }
    }

    sort(files.begin(), files.end());
    for (unsigned int k = 0; k < files.size(); k++)
    {
        cout << files[k].Path() << endl;
    }
}


int main(int argc, char * argv[])
{
    string dirname = ".";
    if (argc > 1)
    {
        dirname = argv[1];
    }

    DirStream d(dirname);
    printAll(d, new AndFilter(new SizeFilter(10), new RegExpFilter("\\.cpp$")));

    return 0;
}
```

```
#include <sstream>
#include <iomanip>
using namespace std;

#include "format.h"
#include "directory.h"

string Format::format(const DirEntry& entry)
{
    return entry.Name();
}

string LongFormat::format(const DirEntry& entry)
{
    ostringstream out;
    out << right << setw(10) << entry.Size();
    out << entry.GetDate() << " " << entry.GetTime();
    out << " " << entry.Name();
    return out.str();
}
```

```
#include <iostream>
#include <string>
#include <fstream>
#include <cstdio>   // for atoi
using namespace std;

#include <getopt.h>

void usage()
{
  cout << "use the right args" << endl;
}

int main(int argc, char * argv[])
{
    struct option long_ops[] = {
        {"file",  required_argument, 0, 'f'},
        {"count", required_argument, 0, 'c'}
    };
    string shortArgs = "f:c:";

    string filename;
    int count = 20;
    extern char * optarg;
    char ch;

    while ( (ch = getopt_long(argc,argv, shortArgs.c_str(),
                              long_ops,NULL)) != -1){

        switch (ch){
           case 'f' :
               filename = optarg;
               cout << "filename is " << filename << endl;
               break;
           case 'c':
               count = atoi(optarg);
               cout << "count is " << count << endl;
               break;
            case '?':
                break;

            default:
              usage();
        }
    }
    if (filename == ""){
        cout << "Using standard input" << endl;
    }
    else {
        cout << "using filename : " << filename << endl;
    }
    cout << "count is " << count << endl;
}
```

```
#include <iostream>
#include <string>
#include <cstdio>   // for atoi
using namespace std;

#include <getopt.h>


#include "filter.h"
#include "sorter.h"
#include "printer.h"
#include "format.h"
#include "oolsprocessor.h"
#include "directory.h"

class NoDotFilter : public Filter
{
  public:

    virtual bool operator()(const DirEntry& entry) const
    {
        return entry.Name()[0] != '.';
    }
};


void usage()
{
  cout << "use the right args" << endl;
  cout << "if this program were more helpful" << endl;
  cout << "it would tell you what those arguments are" << endl;
}


static struct option long_ops[] = {
    {"dir",        required_argument,0,'d'},
    {"size",       no_argument,      0,'s'},
    {"time",       no_argument,      0,'t'},
//    {"extension", no_argument,      0,'X'},
    {"reverse",    no_argument,      0,'r'},
    {"all",        no_argument,      0,'a'},
    {"nodir",      no_argument,      0,'n'},
    {"long",       no_argument,      0,'l'},
    {"ignore-backups", no_argument, 0,'B'},
    {"cols",       required_argument,0,'C'},
    {"ignore",     required_argument,0,'I'},

    {0,0,0,0}
};

static string shortArgs = "d:stranBC:I:";


int main(int argc, char * argv[])
{
    string dirname = ".";


    string filename;
    int cols = 4;
    extern char * optarg;
    char ch;

    Sorter * sorter = new AlphSorter();
    Filter * filter = new NoDotFilter();
    Filter * regexp = new FalseFilter();   // all false
    Printer * printer = new ColumnPrinter();
    Format * format = new Format();
```



```
    bool reverseSort = false;

    while ( (ch = getopt_long(argc,argv, shortArgs.c_str(),
                              long_ops,NULL)) != -1){

        switch (ch){
           case 'C':
               cols = atoi(optarg);
               break;
           case 'd':
               dirname = optarg;
               break;
           case 's':
               sorter = new SizeSorter();
               break;
           case 'r':
               reverseSort = true;
               break;
           case 't':
               sorter = new TimeSorter();
               break;
           case 'a':
               filter = new TrueFilter();
               break;
           case 'I':
               regexp = new RegExpFilter(optarg);
               break;
           case 'l':
               format = new LongFormat();
               break;
        case 'B':
        case 'n':
            cout << ch << " not implemented" << endl;
            break;

           default:
              usage();
        }
    }
    if (reverseSort){
        sorter = new ReverseSorter(sorter);
    }
    regexp = new NotFilter(regexp);
    filter = new AndFilter(filter,regexp);
    printer->setCols(cols);

    OOLSprocessor oolsp;
    oolsp.process(dirname,sorter,printer,format,filter);

}
```

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;

#include "directory.h"
#include "iteradapter.h"

#include "filter.h"
#include "sorter.h"
#include "format.h"
#include "printer.h"

#include "oolsprocessor.h"

typedef iteradapter<DirEntry,DirStream> dirIt;

OOLSprocessor::OOLSprocessor()
    : mySorter(0),
      myFilter(0),
      myFormat(0),
      myPrinter(0)
{

}

void OOLSprocessor::process(const string& dirname,
                            Sorter * sorter,
                            Printer * printer,
                            Format * format,
                            Filter * filter)
{
    DirStream dirs(dirname);
    vector<DirEntry> v(dirIt::begin(dirs), dirIt::end());

    sort(v.begin(), v.end(), *sorter);

    vector<string> filtered;

    for(int k=0; k < v.size(); k++){
        if ( (*filter)(v[k])){
            filtered.push_back(format->format(v[k]));
        }
    }

    printer->print(cout,filtered);

}
```

```
#include "andfilter.h"
#include "directory.h"

OrFilter::OrFilter(Filter * a, Filter * b)
    : myFirst(a),
      mySecond(b)
{

}

bool OrFilter::operator()(const DirEntry& entry) const
{
    return (*myFirst)(entry) || (*mySecond)(entry);
}
```

```
#include <vector>
using namespace std;

#include "printer.h"

static void tab(ostream& out,int n)
{
    for(int k=0; k < n; k++){
        out << ' ';
    }
}


Printer::Printer()
{

}

void Printer::print(ostream& out, vector<string>& list) const
{
    for(int k=0; k < list.size(); k++){
        out << list[k] << endl;
    }
}


ColumnPrinter::ColumnPrinter()
    : myCols(4)
{

}

void ColumnPrinter::setCols(int cols)
{
    myCols = cols;
}

void ColumnPrinter::print(ostream& out, vector<string>& list) const
{
    int size = list.size();
    int rows = size/myCols;
    if (size % myCols != 0) rows++;

    vector<int> max(myCols,0);

    // find max length string in each column

    for(int k=0; k < list.size(); k++){
        int index = k/rows;
        if (list[k].length() > max[index]){
            max[index] = list[k].length();
        }
    }


    for(int k=0; k < rows; k++) {
        for(int j=k; j < list.size(); j += rows){
            int index = j/rows;
            out << list[j];
            tab(out,max[index] - list[j].length() + 2);
        }
        out << endl;
    }
}
```

```
#include <stdexcept>
using namespace std;
#include <libgen.h>     // for regex and regcmp

#include "directory.h"
#include "regexpfilter.h"

RegExpFilter::RegExpFilter(const string& s)
    : myString(s)
{
    char * str = regcmp(myString.c_str(), (char *) 0 );
    if (str == 0) {
        throw new runtime_error(string("error parsing regexp ")+s);
    }
    myString = string(str);
}

bool RegExpFilter::operator()(const DirEntry& entry) const
{
    return regex(myString.c_str(), entry.Name().c_str(), (char *)0) != NULL;
}
```

```
#include "sizefilter.h"
#include "directory.h"

SizeFilter::SizeFilter(int size)
    : mySize(size)
{

}

bool SizeFilter::ok(const DirEntry& entry) const
{
    return entry.Size() >= mySize;
}
```

```
#include "sorter.h"

bool Sorter::operator() (const DirEntry& lhs,
                         const DirEntry& rhs) const
{
    return (*mySorter)(lhs,rhs);
}
```