CSE 165/ENGR 140
Intro to Object Orient Program

Lecture 7 – Data Abstraction
Announcement

- Lab #2
  - Due date: 2/12 (Thursday) at 11:59PM.

- Quiz #1 on 2/19 (Thursday) during lecture

- Reading assignment
  - Ch. 5 & 6
What is the main point behind “data abstraction”?

Wikipedia says:

“In computer science, abstraction is the process by which data and programs are defined with a representation similar in form to its meaning (semantics), while hiding away the implementation details. Abstraction tries to reduce and factor out details so that the programmer can focus on a few concepts at a time. A system can have several abstraction layers whereby different meanings and amounts of detail are exposed to the programmer. For example, low-level abstraction layers expose details of the computer hardware where the program is run, while high-level layers deal with the business logic of the program.”

Simplify data access and use:

Hide details and design appropriate manipulation interface
Recap

- Libraries
  - Class interface
  - Class source code
  - Difference between C and C++
Abstract data type (ADT)

- We can create new data type by packaging data with functions
  - Stash creates a new data type using array
  - Stash has functions to control data (add, fetch, inflate, etc.)
- There are many ADT in the Standard Template Library (STL) of C++.
  - Vectors, lists, stacks, queues, etc.
//: C04:Sizeof.cpp
// Sizes of structs
#include <iostream>
using namespace std;

struct A {
    int i[100];
};

struct B {
    void f();
};

void B::f() {}

int main() {
    cout << "sizeof struct A = " << sizeof(A) << " bytes" << endl;
    cout << "sizeof struct B = " << sizeof(B) << " bytes" << endl;
}
**Nested structures**

```c
struct movies {
    string title;
    int year;
};

struct friends {
    string name;
    string email;
    movies favorite_movie;
};

charlie, maria

friends * p_friends = &charlie;
charlie.name = ...
maria.favorite_movie.title = ...
charlie.favorite_movie.year = ...
p_friends->favorite_movie.title = ...
```
Nested structures: Stack example

- What is a stack of items?
- How are the items in a stack organized?
  - Where do you put a new item (insertion) in a stack?
  - From where do you remove an item (deletion) in a stack?
    - Last-in first-out
- We can also organize our data in stacks.
- Why do we want to use stacks?
  - History of web browser
  - Un-do function of a text editor
  - Matching "{}" or "()" in a cpp editor
Nested structures: Stack example

//: C04:Stack.h
// Nested struct in linked list
#ifndef STACK_H
#define STACK_H

struct Stack {

    struct Link {
        void* data;
        Link* next;
        void initialize(void* dat, Link* nxt);
    }* head;

    void initialize();
    void push(void* dat);
    void* peek();
    void* pop();
    void cleanup();
};

#endif // STACK_H
Stack class

//: C04:Stack.cpp {0}
#include "Stack.h"
#include "../require.h"
using namespace std;

//Stack::Link has only one member method:
void Stack::Link::initialize(void* dat, Link* nxt) {
    data = dat;
    next = nxt;
}
// Stack methods:
void Stack::initialize() { head = 0; }

void Stack::push(void* dat) {
    Link* newLink = new Link;
    newLink->initialize(dat, head);
    head = newLink;
}

void* Stack::peek() {
    require(head != 0, "Stack empty");
    return head->data;
}

void* Stack::pop() {
    if(head == 0) return 0;
    void* result = head->data;
    Link* oldHead = head;
    head = head->next;
    delete oldHead;
    return result;
}
// Stack methods:
void Stack::cleanup() {
    // This implementation does not do anything, it just
    // requires the stack to be empty:
    require(head == 0, "Stack not empty");
}

void Stack::cleanup_notused() {
    // We could do something like empty the stack, BUT
    // we do not know the type of the objects stored in the
    // stack, so we cannot free them...
    while ( pop() ); // works, but may create memory leak...
    require(head == 0, "Stack not empty");
}
Using stack

//: C04:StackTest.cpp
using namespace std;

int main(int argc, char* argv[]) { //Run the program with input arguments
    ifstream in(argv[1]);
    Stack textlines;
    textlines.initialize();
    string line;

    // Read file and store lines in the Stack:
    while(getline(in, line))
        textlines.push(new string(line));

    // Pop the lines from the Stack and print them:
    string* s;
    while((s = (string*)textlines.pop()) != 0) {
        cout << *s << endl;
        delete s;
    }
    textlines.cleanup();
}
We use the global scope resolution operator (::, with nothing in front of it) to select a global identifier.

```cpp
//: C04:Scoperes.cpp
// Global scope resolution

int a;    // 1. Variable a is in the global scope
void f() {} // 2. Function f is in the global scope

struct S {
    int a;    // 3. a is a member of S, its global scope is S::a
    void f(); // 4. f is a member of S, its global scope is S::f
};

void S::f() {
    ::f();    // 5. Would be recursive otherwise!
    ::a++;    // 6. Select the global a
    a--;      // 7. The a at struct scope
}
int main() { S s; f(); s.f(); }
```