PIC 10A

Final Review

Final exam

- Thursday, June 11, 2015 11:30 AM - 2:30 PM MS 5200.
  In our usual class room. (Verify on my.ucla.edu!!)
- The final exam is worth 30% of your grade, same weight as 2 midterms. Could be 50% if grading option 2 turns out better for you.
- Length is also roughly 2 midterms (~8 questions with parts).
- Very similar in style to 2nd midterm.
- Exam covers Ch. 1-7 + Sec 9.1, 14.4
- Remember to bring your student ID.

How to study for the exam

Read the slides.

Work through as many practice exams and problems as you can.

See what you missed on midterms.

Look at your notes and examples your TA did.

Review homework assignments.

Read the book.

Seems like a lot of work? IT IS! Get going.

Syntax

Knowing the syntax also helps you write correct structure for your program.

Knowing the syntax reinforces other concepts.

If you are not confused about syntax the problem becomes clearer.

There is no excuse for not knowing the syntax. All it takes is some memorization.

I/O

Know which way the arrows go with cout and cin!

To read a whole line of input into a string use
`getline(cin,my_string);`

```
<iomanip> stuff
cout << fixed;
cout << setprecision(5) << number;
cout << setw(4) << col1 << setw(4) << col2;
```

You should know about cin buffer and fail state. Recall how we used cin in a while loop.

Good problem to check your knowledge: Homework 2.

Data types and arithmetic

Example: Casting / Integer division

```
int a=5;
int b=7;
double result = (double)(a)/b;
```

The only unusual arithmetic operator is %. Be sure you understand it and know how to use it.

HW #1 should be crystal clear to you now.

Know all the shorthand notation for assignment and incrementation. +=, counter++, etc.

Good problem to check your knowledge: Midterm 1, problem 1

Strings

Remember to `#include <string>`

Concatination: Use a + to append one string to the end of another.

Memorize all the string member functions we talked about and make sure you understand how to use them.

Good problem to check your knowledge: Midterm 1, problem 5.

Control structures

You should definitely know the syntax for all control structures.

You should be able to convert one type of loop to another.

You should know when to use a for loop and when to use a while loop.

You should be very comfortable with &&, || type expressions.

You should understand booleans.

Good homework to look at to review this stuff is the blackjack program

Functions

Know the syntax for function prototype, function definition and for calling a function.

It is quite likely that when I ask you to write code that does some task I will ask you to write a function that does the task.

You should know what a void function is.

You should know how to write a function returning type bool and know how and when to use such a function.

You should know the difference between passing by value and passing by reference.

Work through some examples where you chase values of variables like we did in class.

Function definition

Write a function called divide_by_2 that takes as an argument a vector of ints and returns a vector where all even elements are divided by 2. If the vector is:

5	2	8	3	12

The returned vector should be:

5	1	4	3	6

```
vector<int> divide_by_2(vector<int> v)
{
    for(int i=0; i<v.size(); i++)
    {
        if (v[i] %2 ==0)
            v[i] /=2;
    }
    return v;
}
```

Function prototype

Just take the first line of the function definition and add a ; to the end.

```
#include<vector>
using namespace std;
vector<int> divide_by_2(vector<int> v);
```

Calling a function

Never put any types in function call!!

Function call only has names of variables.

```
vector<int> initial;

// Somehow we fill initial vector

vector<int> final = divide_by_2(initial);
```

Scope

You should know the scope rules that we discussed in class.

What is a life span of a variable?

What does local scope mean?

More “inner” variables always supersede the “outer” variables.

Know what global variables are and how to declare them and when they are appropriate.

Understand what happens if you declare a variable inside a loop.

Review the “massive scope example”.

Passing by value vs. Passing by reference

- You should thoroughly understand the difference between the two.
- You should also know when to pass by reference:
- Passing a class to a function
- Passing a vector to a function

Example: Swap function

```
void swap(int &a, int &b)
{
    int temp;
    temp = a;
    a=b;
    b=temp;
}
```

Passing by value vs. Passing by reference

Search through a vector and return the first position where value occurs.

```
int find(const vector<int> &list, int value)
{
    int i = 0;

    while (i < list.size() && list[i] != value)
    {
            i++;
    }
    return i;
}
```

Here we use passing by reference for efficiency

Passing by value vs. Passing by reference

Search through a vector and return the first position where value occurs.

```
void foo(Employee &e)
{
    e.set_salary(1500);
}
```

If we want the change to e be recorded in main we must pass e by reference.

Random numbers

You should know how and why to seed a random number generator.

```
srand((int)(time(0));
```

You should know how to get random numbers in a specified range.

The magic formula to get a number in a range [a,b] is

```
a + rand() % (b-a+1)
```

but you should know how to adapt this formula if need arises.

Good example of this is the question: Write a single line of code that assigns to an `int x` a random even number in the range [50,100].

```
// Get a number in range 25-50 then multiply by 2.
int x = 2 * (rand() % (50-25+1) + 25);
```

Classes

Know the syntax!

Declaring a class.

Implementing a class.

Know how to write a constructor, both default and regular.

Know how to overload an operator.

Class declaration

A class called `Team` has the following data: `name, wins, losses. name` is type string and `wins` and `losses` are type int. The class has a default constructor which sets name to "" and `wins` and `losses` to 0 and a regular constructor that takes as arguments a `name, wins` and `losses`. It has an accessor `get_name` that returns the `name`. It also has a mutator called `defeats` that takes as an argument another `Team` object and when called, it increments `wins` by one for the instance object and increments `losses` for the argument object by one. It has an operator > that compares the ratio of `wins` to `losses`.

An example of the use of class is:

```
Team A("Isotopes", 2, 14);
Team B("Shelbyvillians", 6, 5);

B.defeats(A);  //B's wins are incremented and A's losses are
incremented.

if (A > B)
    cout << A.get_name() << " is the better team.";
else
    cout << B.get_name() << " is the better team.";
```

Class implementation

```
class Team
{
public:
    Team();
    Team(string new_name, int new_wins, int new_losses);
    void defeats(Team &T);
    bool operator > (Team T) const;
    string get_name() const;
    int get_wins() const;
    int get_losses() const;
private:
    string name;
    int wins;
    int losses;
};
```

Class implementation

```
Team::Team()
{
    name="";
    wins = 0;
    losses = 0;
}

Team::Team(string new_name, int new_wins, int new_losses)
{
    name = new_name;
    wins = new_wins;
    losses = new_losses;
}
```

Class implementation

```
void Team::defeats(Team &T)
{
    wins++;
    T.losses++;
}

bool Team::operator > (Team T) const
{
    if (losses==0)
        return true;
    if (T.losses == 0)
        return false;
    if ((double)wins/losses > (double) T.wins / T.losses)
        return true;
    else
        return false;
}

string Team::get_name() const
{
    return name;
}
```

Vectors and Arrays

Both arrays and vectors have data of same type.

Access elements with array_name[i] or vector_name[i].

Both have indexes that go from 0 to size-1.

Array size has to be known at compilation time.

Vector size can change because of push_back and pop_back and resize member functions.

Arrays have no member functions.

Arrays are always passed by reference automatically.

Vectors are passed by value by default, but you can pass by reference if you want.

Vectors and Arrays

A function cannot return an array, but a function can return a vector.

2-D arrays are much easier to handle than a 2-D vector.

Arrays are a primitive C++ construct and are more efficient than vectors.

With an array you need to have your own variable that tracks the size of the array. Often this variable needs to be passed to a function with the array.

Dynamic arrays will not be in the final

Vectors and Arrays

There will without a doubt be questions dealing with arrays and vectors in the final.

You should know how to traverse through a vector or an array.

You should also know how to go backwards through a vector or an array.

We discussed a lot of important vector/array functions in class. You should be familiar with all of them.

Review the phone book homework assignment.

Vectors and Arrays

Write a bool function that takes as argument two vectors of strings (v1 and v2) and returns true if v2 contains all elements of v1 in some order and ignoring multiplicities.

```
bool same_elements(vector<string> v1, vector<string> v2)
{
    bool found=false;
    for (int i=0; i<v1.size(); i++)
    {
        for (int j=0; j<v2.size(); j++)
        {
            if (v1[i] == v2[j])
                found = true;
        }
        if (found)
            found = false;
        else
            return false;
    }
    return true;
}
```

2-D Arrays

Write a snippet of code that declares a 10x10, 2-D array of doubles and then fills it in a following way. For each entry randomly either make the entry equal to its column number divided by row number or row number divided by column number. We have to be careful not to divide by 0, so if the denominator is 0 just change the denominator to 1.

2D-Arrays cont.

```
srand(time(0));
double a[10][10];

for (int i=0; i < 10; i++)
{
     for (int j = 0; j < 10; j++)
     {
          if (rand() % 2)
          {
               if (j !=0)
                    a[i][j] = double(i)/j;
               else
                    a[i][j] = i;
          }
          else
          {
               if (i !=0)
                    a[i][j] = double(j)/i;
               else
                    a[i][j] = j;
          }
     }
}
```

2D-Arrays cont.

Output the array:

```
cout << setprecision(2) << fixed;

for (int i=0; i < 10; i++)
{
    for (int j = 0; j < 10; j++)
    {

    cout << setw(5) << a[i][j];

    }
    cout << "\n";
}
```

File I/O

Once you have your file objects fin and fout and have the file open, reading and writing to a file is just like reading and writing to a console.

You can even use getline(fin, my_string); and all the <iomanip> stuff.

So that leaves only 3 things to know:

1)How do I declare fin, fout objects.
2)How do I open a file for reading or writing.
3)How do I pass the fin and fout objects to functions

File I/O

We need to `#include <fstream>`

```
int main()
{
ifstream fin;
ofstream fout;

string input_file_name;

cin >> input_file_name;

fin.open(input_file_name.c_str());
fout.open("output.txt");

...etc....
```

File I/O

When we use a function to write to a file we should pass our file stream to the function. Otherwise the function does not know what fout or fin mean!

```
void output (ofstream &fout, ...other arguments...)
{
// file stuff
}

void input (ifstream &fout , ...other arguments...)
{
// file stuff
}
```

The streams should be passed by reference. This is so that we don't lose our place in the file. Remember that fin and fout are keeping track of where we are in the file. If we pass by value then the next time we come to the function we will have lost our place in the file.

File I/O

Write a full program starting from `#include` that accomplishes the following task. Ask the user to enter a filename, then open the file for reading. Next, write the contents of the file in reverse order into a file called `backwards.txt`. You may assume that the file that the user specifies contains text separated by spaces. For example, if the file contains the phrase:

Hello! How's it going?

The file `backwards.txt` should have the content

going? it How's Hello!

Hint: First read the contents of the file that the user specifies into a vector.

File I/O

```
#include<iostream>
#include<string>
#include<vector>
#include<fstream>

using namespace std;

int main()
{

     string filename;
     ofstream fout;
     ifstream fin;
     string word;

     vector<string> file_content;

     cout << "Please enter a file name: ";

     cin >> filename;

     fin.open(filename.c_str());
```

Continued on next slide...

File I/O

```
    while (fin >> word)
    {
        file_content.push_back(word);
    }

    fout.open("backwards.txt");

    for (int i=file_content.size()-1; i >=0; i--)
    {
        cout << file_content[i];
        fout << file_content[i] << " ";
    }

    fout.close(); // optional

    return 0;
}
```

Pointers

We create a pointer to block of given type with the declaration:

```
type* pointer_name;
```

A pointer stores a memory address.

```
double x = 3.14;
double* p = &x;
```

Alternatively, you can allocate a blank memory block with new.

```
Person *P = new Person;
```

Dereferencing accesses the pointee's value directly with *.

```
*p = -42.3;
```

If you set a pointer equal to array name, it points to 1st element.

```
string Simpsons[5] = {"Homer","Marge","Bart","Lisa","Maggie"};
string* s = Simpsons; //s points to 1st string Homer.
```

Adding to the pointer moves the pointer through the array.

```
s = s+2; //Now s points to 3rd element Bart
```

Pointers

Often we will have a pointer to a class

```
Point *p = new Point(3.2,5.0);
```

We can access the class by dereferencing the pointer.

```
cout << "X value is: " << (*p).get_x();
```

But all the cool kids use the arrow method:

```
cout << "X value is: " << p->get_x();
```

Things can look complicated but if you track data types its easy:

```
people[i]->get_best_friend()->get_name();
```

Pointers

```
void fun (int* p, int *&q)
{
    *p = 100;
    p += 2;
    q = p;
    *q = *(p-1)+*(p+1);
    cout<<*p<<" "<<*q<<"\n";
    return;
}

int main()
{
    int x[4] = {1,2,3,4};
    int *p1 = x;
    int *p2;
    fun(p1,p2);
    for(int i=0; i<4; i++)
        cout<<x[i]<<" ";
    cout<<"\n"<<*p1<<*p2;
    return 0;
}
```

Pointers

```
#include <iostream>
using namespace std;
int fun ( int*& p, int* q) {
    *p = 12;
    p = p+3;
    q = p-1;
    *q = *(p+1) + *q;
    *p += 2;
    cout << "p=" << *p << " q=" << *q << "\n";
    return (*q)%(*p);
}
int main( ) {
    int y[5] = {1,2,3,4,5};
    int* p = y;
    int *q = new int;
    *q = fun ( p, q );
    for (int i=0; i<5; i++)
        cout << y[i] << " ";
    cout << "\np=" << *p << " q=" << *q;
    return 0;
}
```

Pointers

7. [20pts] Pointer chasing

What is the output of the following code?

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

void foo (int* &c, int* d)
{
	*(c+2) = 10;
	int *p = d;
	*p = 7;
	d = c;
	*d = *c + *(c-1);
	c++; // :)
	*c = 0;
}

int main()
{
	int a[5]={1,2,3,4,5};
	int *p = &a[1];
	int *q = new int;
	*q = *(a+4);
	cout << *p <<" " << *q <<"\n";
	foo (p,q);
	cout << *p <<" " << *q <<"\n";
	for (int i=0; i < 5; i++)
		cout << a[i] <<" ";
}
```