CS 368-1 :: C++ for Java Programmers :: Lecture 6 Be sure to sign the attendance sheet!

A. Preview

Last Time (Feb 25)	**Today (Mar 4)**	**Next Time (Mar 11)**
• Outside reading: Linked Lists • **typedef** • **void** as a type • review C++ memory model • pointers as return types • pointers to functions • arrays of pointers • pointers to pointers	start Chapter 4 • defining classes in C++ • .h and .cpp files • ifndef …..define • private is the default • const for accessor functions • #include statements • constructors	continue Chapter 4 • initializer lists • multi-file compilation • parameter default values • **`explicit`** • makefiles • Setting up the Big 3: • copy constructor • destructor • copy assignment

B. Announcements:

1. Homework and Program 1 are in the process of being graded. Your grades should be available by March 11.

2. Outside Reading: Linked Lists: To complete your homework assignment, and also to complete program2, you should review your 367 knowledge of Linked Lists by reading this file. If you don't have a strong grasp of linked lists, you might want to consider a program partner who has a strong understanding.

3. Program p2 is due Thursday March 27th at 8:00 PM. You will know enough to complete this program before spring break if you want, but it will be due after spring break to allow students more flexibility. Be aware that program p3 will probably be due 10 days after p2.

C. What are the benefits of Classes? Classes allow us to……

1. Encapsulate information (bundle data and functionality, restrict access)
2. Reuse code in other programs
3. Make large software projects easier to design, debug, and maintain …..usually :)

General Comment: Remember that C++ likes to

give the programmer more than one way to do something
treat everything like a primitive type
use **const** in a lot of different ways

To see how this works, we are going to create a class that models a song Play List

D. Classes in C++ can have a public interface, called the header file, which acts as class declaration. (It can also be called the class specification or class interface.) Header files are optional in C++ but we expect you to always use them.

# is a symbol called a preprocessor command

We use **`#ifndef…#define`**

because it allows one class to be used in multiple classes without worrying about that class being defined more than once

read more….

```
//  PlayList.h
//  Lecture06_inclass
//  Created by Andrew Kuemmel on 3/4/14.

#ifndef __Lecture06_inclass__PlayList__
#define __Lecture06_inclass__PlayList__

#include <iostream>
using namespace std;  // defines string

struct Song {
    string name;
    string artist;
};
```

.h file can include structs associated with this class or other typdefs

```
struct SongNode{
    Song s;
    SongNode* next;
};

class PlayList{
```

Java assumes everything is public, but **C++ assumes everything is private**

“private” declaration is optional but expected in 368

```
private:
    string nameOfList;
    int numSongs;
    SongNode* head;
    SongNode* tail;
```

public is required if you want to make a member function public

```
public:
    PlayList();
    PlayList(string name);
```

`const` means that this function will not change the data members of the class

```
    // accessor member function
    void print() const;
    int countArtist(string artistName) const;

    // mutator member function
    void addToEnd(Song newSong);
    void changeName(string newName);
    void changePos(Song s, int newPos);
```

don’t forget that you need to end your class declaration with a semi-colon !!!

```
};

#endif /* defined(__Lecture06_inclass__PlayList__) */
```

E. At this step, I like to write the main function to see if our interface needs changing

- we write it in a separate file (anything.cpp)
- we #include the.h file
- we can declare variables of anything that is named in the .h file
- we can instantiate objects of type PlayList from the stack or the heap

```
//  main.cpp
//  Lecture06_inclass
//  Created by Andrew Kuemmel on 3/4/14.

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

int main()
{
    PlayList list0 ("list0");
    PlayList list1 ("list1");
    list1.addToEnd({"Firework", "Katy Perry"});
    list1.print();

    PlayList* list2 = new PlayList("list2");
    list2->addToEnd({"May it Be", "Enya"});
    list2->print();

    list0 =list1;
    // this makes a "shallow copy" of list1
    list0.changeName("LisT ZERO");
    list0.print();
    list1.print();

    delete list2; // shallow delete
    return 0;
}
```

F. Classes have a source file (.cpp) which separates the implementation from the declaration

- we need to #include “PlayList.h”.....this literally copies and pastes that code into this file
- constructors…..in C++ you can’t have one constructor call another
    but C++ gives other options ………..
- if your class has no constructor written, C++ makes a default one
    but if your class has at least one constructor, then C++ will not make a default
    you should always write a default constructor to avoid updating errors
- The symbol :: is called a “scope resolution operator” or “scope operator”

```
//  PlayList.cpp
//  Lecture06_inclass
//  Created by Andrew Kuemmel on 3/4/14.

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

PlayList::PlayList(){
    nameOfList = "Untitled";
    numSongs   = 0;
    head =  NULL;
    tail = NULL;
}

PlayList::PlayList(string name){
    nameOfList = name;
    numSongs   = 0;
    head =  NULL;
    tail = NULL;
}

void PlayList::changeName(string n){
    nameOfList = n;
}

void PlayList::print() const{
    SongNode* temp = head;
    cout << nameOfList<< " [ ";
    while (temp != NULL) {
        cout << temp->s.name << " by " << temp->s.artist << ", " ;
        temp = temp->next;
    }
    cout << " ] has " << numSongs << " songs." << endl;
}

void PlayList::addToEnd(Song newSong){
    SongNode* temp = new SongNode;
    temp->s = newSong;
    temp->next = NULL;

    if (head == NULL){
        head = temp;
    }
    else{
        tail->next = temp;
    }
    tail = temp;
    numSongs++;
}
```