Final Review

Lecture 15

Format

Part 1 (50%)

- Multiple choice
- Predict the output

Notes

- One 8.5x11” (front/back) page of notes
- All responses in pen
- No calculators, books, computers, phones, etc.

Part 2 (50%)

- 2 programming problems

Notes

- One 8.5x11” (front/back) page of notes
- Submission via Blackboard, zipped source only
- No calculators, books, phones
- No Internet resources

Content

Everything, including…

- All of COMP128
- Strings (C strings and **`string`** class)
- Command line arguments
- Vectors
- Pointers, dynamic arrays
- Structures, Classes, **`friend`** functions/classes, **`const`** correctness
- Operator overloading
  - Assignment, extraction/insertion
- Code libraries/separate compilation
  - Headers, include guards, type definitions, namespaces
- Linked lists, stacks/queues
- Recursion
  - Base case, recursive step
- Inheritance, polymorphism
  - [Pure] virtual functions
- C++ kitchen sink
  - Exceptions, iterators, rng, casting, enumeration, pairs
  - Deep vs. shallow copy, big three, copy constructor

Strings

- C strings vs. **`string`** class
- Relevant libraries
- Declaration, initialization, accessing characters
- Common functions (e.g. length, concatenation, comparison, I/O)

Command Line Arguments

- **`argv`**, **`argc`**
  - Data types, meaning

Vectors

- **`size`** vs. **`capacity`**
- **`at`** vs. **`[]`**
- **`push_back`** and automatic initialization

Pointers, Dynamic Arrays

- Declaration, `*`, `&`
- Static vs. dynamic allocation
  - Stack vs. heap, memory leak
  - `NULL`, `new`, `delete`
- Pointer-Array duality
  - `(ptr+i) = &arr[i]`, `*(ptr+i) = arr[i]`
- Dynamic arrays
  - `new`, `delete[]`
- Multi-dimensional arrays

Structures, Classes, `friend`, `const`

- Syntax
  - Members, access levels, function definitions
  - Declaration, initialization, access
  - Pointer access: `->` `(*o).`
  - Meaning/when to use: `this`
- Who can access what
- Encapsulation, information hiding
- Constructors (default), destructors
- Multiple meanings of `const`

Operator Overloading

- General syntax
- Automatic type conversion
- Binary, unary, extraction/insertion, assignment
- Relationship to `friend`

Code Libraries

- How to separate code into multiple files
  - Remember include guards!
- What does `#include` do?
- What does a type definition do?
- What is a namespace?
  - Different usage: `using` vs. `::` vs. `{}`
  - Common namespaces (e.g. `std`, global)

Exercise

- Define the member functions for the Car class.
- Put it within the **`transportation`** namespace.
- Convert the following code to a proper code library + application file using three separate units (**`Car.h`**, **`Car.cpp`**, **`main.cpp`**).

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

class Car
{
public:
	Car(string make, string model);
	string getMake() const;
	string getModel() const;

	friend ostream& operator <<(ostream&
		outs, const Car& c);

private:
	string make;
	string model;
};

int main()
{
	Car c1( "Toyota", "Prius" );
	cout << c1 << endl;
	// The Toyota Prius rocks!
}
```

Answer

Car.h

```
#ifndef __CAR_H
#define __CAR_H

#include <string>
#include <ostream>

namespace transportation
{

class Car
{
public:
        Car(std::string make, std::string model);
        std::string getMake() const;
        std::string getModel() const;

        friend std::ostream& operator
        <<(std::ostream& outs, const Car& c);

private:
        std::string make;
        std::string model;
};

}

#endif
```

Car.cpp

```
#include "Car.h"

namespace transportation
{

Car::Car(std::string make, std::string model):
make(make), model(model) {}

std::string Car::getMake() const
{
        return make;
}

std::string Car::getModel() const
{
        return model;
}

std::ostream& operator <<(std::ostream& outs,
                          const Car& c)
{
        outs << "The " << c.make
             << " " << c.model << " rocks!";
        return outs;
}

}
```

main.cpp

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

int main()
{
        transportation::Car c1( "Toyota", "Prius" );
        cout << c1 << endl;
        // The Toyota Prius rocks!
}
```

Linked Lists

- How to build/modify/use/deallocate
- Purpose of pointers
- Relationship to stacks/queues
  - Basic operations
  - How to implement via LL

Exercise

Write a function to add a node to the beginning of a linked list of characters. Then add the characters to spell the name of your program using **`argv`** (and write a function to print the contents of the list). Finally, write a function to deallocate the list.

Answer (1)

```
struct Node
{
    char letter;
    Node* next;
};

void add(Node*& head, char letter)
{
    Node* temp = new Node;
    temp->next = head;
    temp->letter = letter;
    head = temp;
}

void show(Node* head)
{
    for ( Node* n=head; n!=NULL; n=n->next )
        cout << n->letter;
    cout << endl;
}
```

Answer (2)

```
void deallocate(Node*& head)
{
    while ( head != NULL )
    {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

int main(int argc, const char* argv[])
{
    Node* head = NULL;

    for ( int i=strlen( argv[0] )-1; i>=0; i-- )
        add( head, argv[0][i] );
    show( head );
    deallocate( head );

    return 0;
}
```

Recursion

- How to execute a recursive function
- How to write a recursive function
- Meanings: base case, recursive step

Exercise

Examine the following sequence of numbers and determine its pattern. Then write a C++ function to recursively generate any number in the sequence.

7, 15, 32, 67, 138, …

Answer

```
int f(int n)
{
  if ( n == 0 )
    return 7;
  else
    return 2*f(n-1) + n;
}
```

Inheritance

- Meaning: inheritance, polymorphism
- How to make a derived class
  - How to use polymorphically
- Access levels: `protected`
- Constructor execution ordering
  - Initializer lists
- Late binding via `virtual`
  - Abstract classes

Exercise

Finish the code to make this program execute as shown.

```
class Diplomat
{
public:
        Diplomat(string country): country(country) {}
        friend ostream& operator <<(ostream& outs, const Diplomat& d)
        {
                outs << d.sayHi() << " from " << d.country;
                return outs;
        }

protected:
        virtual string sayHi() const = 0;

private:
        string country;
};

// your code here

int main(int argc, const char* argv[])
{
        vector<Diplomat*> delegation;
        delegation.push_back( new AmericanDiplomat() ); // Howdy from The United States
        delegation.push_back( new BritishDiplomat() ); // Hello from Great Britain
        delegation.push_back( new FrenchDiplomat() ); // Bonjour from France

        for ( vector<Diplomat*>::iterator it=delegation.begin();
              it!=delegation.end();
              it++ )
        {
                cout << *(*it) << endl;
                delete *it;
        }
        delegation.clear();
        return 0;
}
```

Answer

```
class AmericanDiplomat: public Diplomat
{
public:
     AmericanDiplomat(): Diplomat("The United States") {}
protected:
     virtual string sayHi() const { return "Howdy"; }
};

class BritishDiplomat: public Diplomat
{
public:
     BritishDiplomat(): Diplomat("Great Britain") {}
protected:
     virtual string sayHi() const { return "Hello"; }
};

class FrenchDiplomat: public Diplomat
{
public:
     FrenchDiplomat(): Diplomat("France") {}
protected:
     virtual string sayHi() const { return "Bonjour"; }
};
```

C++ Kitchen Sink (1)

- Exceptions: `throws`, `try`/`catch`
  - What can you throw? Why?
- Purpose of iterators?
  - Basic usage
- RNG: purpose?
  - Function of a seed? Basic usage
- Casting: purpose?
  - Basic usage
- Enumeration: purpose?
  - Basic usage

C++ Kitchen Sink (2)

- Pairs: purpose?
  - Basic usage
- Deep vs. shallow copy
  - Big Three
  - Result w.r.t. memory

Wrap Up

- You have now had exposure to most of the beginner-moderate features of C++
  - Much of these carry to many other languages
- You have also had a taste of computer science data structures (e.g. Linked List, Stack, Queue) and programming paradigms (e.g. recursion, OOP)
- Thank you for working super hard this semester :-)