Midterm Review

Lecture 9

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%)

- 3 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
- Operator overloading
- Separate compilation, **`const`** correctness

Strings

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

Exercise

Write two functions (one for C strings named **`cstr_long`**, the other for the **`string`** class, named **`str_long`**) that returns the first character of the longer of two strings supplied as arguments (in the case of a tie, use the first argument; if both are empty, return the null character).

Answer

```
char cstr_long(char str1[],
               char str2[])
{
    int len1 = strlen( str1 );
    int len2 = strlen( str2 );

    if ( len1 == 0 && len2 == 0 )
    {
        return '\0';
    }
    else if ( len1 >= len2 )
    {
        return str1[0];
    }
    else
    {
        return str2[0];
    }
}
```

```
char str_long(const string& str1,
              const string& str2)
{
    if ( str1.length() == 0 &&
         str2.length() == 0 )
    {
        return '\0';
    }
    else if ( str1.length() >=
              str2.length() )
    {
        return str1[0];
    }
    else
    {
        return str2[0];
    }
}
```

Command Line Arguments

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

Exercise

What is the value of **`argc`** and **`argv`** given the following program call

```
> prog abc 1 2 3
```

Answer

```
argc = 5
argv[0] = "prog";
argv[1] = "abc";
argv[2] = "1";
argv[3] = "2";
argv[4] = "3";
```

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

Exercise

Write a function named **`pos_print`** that takes as an argument a pointer to an integer. Treat the pointer as an array and output each value until reaching a negative number. Do not use the `[]` operator.

Answer

```
void pos_print(int* p)
{
   for ( int i=0; *(p+i)>=0; i++ )
   {
      cout << *(p+i) << endl;
   }
}

int main()
{
   int arr[] = { 1, 3, 5, 7, -1 };
   pos_print( arr );
   return 0;
}
```

Structures, Classes, `friend`

- Syntax
  - Members, access levels, function definitions
  - Declaration, initialization, access
  - Meaning/when to use: `this`
- Who can access what
- Encapsulation, information hiding
- Constructors (default), destructors

Operator Overloading

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

Exercise

Create a class representing a sphere. Add a single member variable, **`radius`**. Add member functions to your circle class: **`getRadius()`** should return the value of the member variable, **`setRadius()`** should allow client code to set a valid radius (>0), **`surfaceArea()`** should return the surface area of the sphere, and **`volume()`** should return the volume of the sphere. Add a constructor that takes as an argument the initial radius – if it isn't valid, default to 1. Also add a default constructor that sets the radius to 1. Also overload the insertion operator to output the radius, surface area, and volume. Write a **`main`** function to test the class. Make sure the class satisfies *const correctness*.

Surface area: $4\pi r^2$

Volume: $\frac{4}{3}\pi r^3$

Answer

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

class Sphere
{
public:
	double getRadius() const;
	double surfaceArea() const;
	double volume() const;
	void setRadius(double radius);
	Sphere();
	Sphere(double radius);
	friend ostream& operator <<(ostream& outs,
	                            const Sphere& s);
private:
	double radius;
};

double Sphere::getRadius() const
{
	return radius;
}

double Sphere::surfaceArea() const
{
	return ( 4.0*3.14159*radius*radius );
}

double Sphere::volume() const
{
	return ( (4.0/3.0)*3.14159*radius*radius*radius );
}
```

```
void Sphere::setRadius(double radius)
{
	if ( radius > 0 )
		this->radius = radius;
}

Sphere::Sphere()
{
	radius = 1;
}

Sphere::Sphere(double radius)
{
	if ( radius > 0 )
		this->radius = radius;
	else
		this->radius = 1;
}

ostream& operator <<(ostream& outs, const Sphere& s)
{
	outs << "Radius: " << s.radius << endl
	     << "Surface Area: " << s.surfaceArea() << endl
	     << "Volume: " << s.volume();
	return outs;
}

int main()
{
	Sphere s( 4 );
	cout << s;
	return 0;
}
```

And the Rest…

- Separate compilation
  - Header, include guards, forward declaration

- `const` Correctness
  - Uses of `const`
  - Requirements/promises made