Midterm Review

CS 246

Outline

Linux

Linux Commands

- You will be given the linux command reference for the midterm
    - This means that you do not need to memorize the commands but being familar with them and their options will be helpful
- Important commands to know: cd, ls, wc, egrep, diff, echo

Regular Expressions

You should know the special characters for Regular expressions

- `.` - any one character
- `^` - beginning of a line
- `$` - end of a line
- `*` - zero or more of the previous pattern
- `+` - one or more of the previous pattern
- `?` - zero or one of the previous pattern

Midterm Review

- (expr1 | expr2 | ...) - matches any one of the given expressions
- `[...]` - one occurrence any one of the characters listed
- `[^...]` - one occurrence any one of the characters not listed
  - The last two can be used with ranges such as `A-Z,a-z,0-9`

Special characters can be escaped using \

Midterm Review

Examples of regular expression questions:

- Return every line that contains cs246 or CS246
- Return every line that has a the number 4 eventually followed by a 2
- Return every line which contains only a commented out message to cerr

Globbing Patterns

Globbing patterns can be used with the shell to return multiple file names. When this happens, the glob is expanded by the shell. Special globbing characters include:

- `*` - match any number of any char
- `?` - match one occurrence of any char
- `{expr1,expr2}` - matches one of the expressions
- `[...]` - matches one of the chars
- `[!...]` - matches anything but one of the chars

I/O Redirection

Programs have three built in streams: input (from the keyboard), output and error (to the terminal).

Redirection is used to change where a program's streams receive or send data.

```
./myprog <input.txt >output.txt 2>error.txt
```

To suppress output and error, they can be sent to `/dev/null`.

Midterm Review

Output can be directed through multiple programs using pipelines.

```
egrep "^(...)$" /usr/share/dict/words | egrep "[az]" | wc -w
```

Bash Scripting

A bash script is a series of programs call which can be executed.

Important features of bash scripts:

- if statements
- while loops
- for loops
- variables

Midterm Review

Some useful functions are:

- `-e, -r, -x, -w, -d` - which check various permissions of the following filename
- `-eq, -gt, -ge, -lt, -le, -ne` - compare two integers
- `==, !=` - compare two strings

Midterm Review

```bash
#!/bin/bash

while read line; do
    count=0
    for word in ${line}; do
        count=$(( $count + 1 ))
        var=$(( $count % 1))
        if [ $var -eq 0 ]; then
            echo -n $word
        fi
    done
    echo ""
done < file.txt
```

C++

You are expected to know about the format of if statements, for loops, while loops, pointers, int, bool and basic C from CS 136.

Strings

- Must #include <string>
- Have methods including
  - `c_str()` - returns C style string
  - `length()` - returns length of string
  - `substr(n,m)` - returns m characters starting at the nth char

Streams

I/O Streams

- Must #include <iostream>
- There are 3 basic iostreams: cin, cout, cerr which correspond to stdin, stdout and stderr respectively.
- `endl` can be used to print a newline character to an output or error stream

String Streams

- Must #include <sstream>
- This includes stringstream, istringstream (input) and ostringstream (output)
  - A stringstream can be used as an istringstream and an ostringstream at the same time
- A stringstream can be initialized from a string:

```
string s = "Hello World";
stringstream ss(s);
stringstream st("1 2 3 4 5 6");
```

Midterm Review

- The method str() returns the current contents of any stringstream as a string

File Streams

- Must #include <fstream>
- This includes filestream, ifilestream (input), and ofilestream (output)
- All must be initialized with a C style string
- A file stream can be initialized:

```
string file = "myfile.out";
ifilestream input("myfile.in");
ofilestream output("myfile.out");
```

- If the file for output does not exist, it will be created.
- If the file for output does exist, it will be over written.

Input Stream Operators

The following is a list of useful input stream operators.

- `stream >> var`: attempts to read from stream to var
- `stream.fail()`: returns true if a read from stream has failed
- `stream.clear()`: sets fail bit of stream to false
- `stream.ignore()`: ignores the next char from stream
- `stream.peek()`: returns the next char in stream, does not remove from the char from stream
- `stream.get()`: reads the next char from a file

Midterm Review

- `getline(stream, string)`: reads the remainder of the current line into string

Dynamic Memory

In C++, memory on the heap is allocated using new and deallocated using delete.

```
MyClass * cp = new MyClass;
delete cp;
cp = new MyClass[20];
delete[] cp;
```

- delete[] must be used to deallocate an array on the heap

Pointers

A pointer is a variable which contains a memory address. A pointer can hold the memory address of an individual object or the address of the first element of an array.

Examples:

```
int n = 42;
int * np = 0;                    // NULL int pointer
np = &n;
int * np2 = new int;             // pointer to an int on the heap

// pointer an array of strings on the heap
string * sp = new string[10];
```

References

- A reference is a similar to a pointer except:
  - The address a reference refers to cannot be changed.
  - A reference does not need to be dereferenced
  - A reference cannot be uninitialized
  - A reference cannot be null
- It is better to pass-by-reference than pass-by-value because the parameter is not being copied and doesn't take up more space in memory.

Default Parameters

- A function can be given a parameter which it will automatically use if no parameter is given

```
void function(int x, int y = 0);

function(5);     // runs function with x = 5, y = 0
function(4,2);   // runs function with x = 4, y = 2
```

- default parameters must always be the last parameters listed in a function

Overloading

- We can have multiple functions that have the same name using overloading
- For an overload to be valid, the versions of the function must not be able to be called in the same way. Functions cannot be overloaded on just return type.

Midterm Review

```
int foo(int x);
```

```
int foo(string x);          // valid
char foo(int x);            // invalid
int foo(int y);             // invalid
int foo(int x, int y = 10); // invalid
bool foo(double z);         // valid
char foo(int x, int y);     // valid
```

Operator Overloading

- Built in operators can also be overloaded
- Operators include

```
operator+
operator-
operator*
operator/
operator<<
operator>>
```

- Each of these (and many more) can be overloaded for any user created class.

Classes

- A class is a group of related data and methods

```
struct Rational{
        int numer, denom;

        Rational();
        Rational(int x, int y);
        Rational operator+(const Rational &r);
};
```

Methods

- Methods are functions which are built into a class.
- They are called `var.method()` if var is an object or `var->method()` if var is a pointed to an object.
- Methods have an implicit pointer called *this* to the object they were called on.

Constructors

- Special methods used for creating objects of a class.
- The compiler gives you a default constructor and a C style constructor automatically.

  ```
  MyClass c;
  MyClass c = {1,2,3};
  ```

- Both of these constructors are lost if a new constructor is defined.

Midterm Review

```
struct MyClass{
    int x,y,z;

    // constructor with three fields
    MyClass(int x = 0, int y = 0, int z = 0);
};
```

Initialization List

If we have an object with fields that are const or references, we must initialize them with the initialization list. This is because these variables must be initialized when they are declared.

When an object is initialized, the following steps take place in order:

1. Allocate space
2. Initialize members to default values/initialization list values (in the order they were declared in)
3. Execute constructor body

Copy Constructor

- The copy constructor is a special constructor for initializing an object during initialization.
- Every class is given a copy constructor from the complier.
- If the class is non-contiguous, a copy constructor usually needs to be defined by the user.

```
struct IntClass{
        int * data;

        IntClass(IntClass &n): data(new int(*(n.data))){}
};
```

Midterm Review

This performs a deep copy of the object.

Destructor

- The destructor is the method called when an object is deleted through being popped off the stack or having delete called on it.
- If the class is non-contiguous, a destructor usually needs to be defined by the user.

```
~IntClass(){
        delete data;
}
```

Assignment Operator

- The Assignment Operator is operator=.

```
int x = 6, y =7;
x = y;     // call to the assignment operator.
```

- If the class is non-contiguous, an assignment operator usually needs to be defined by the user.

Midterm Review

```
IntClass operator=(IntClass &n){
        if (this == &n)
                return this;
        IntClass temp;
        temp.data = this->data();
        this->data = new int(*(n.data()));
        delete temp;

        return *this;
}
```

The Rule of Three

If you need to create a copy constructor, destructor or assignment operator, you likely have to create all three?

Why? Because you typically have to create each if the class contains non-contiguous.

Copy and Swap Idiom

If we do have to redefine the copy constructor, destructor and assignment operator, we might as well use the copy constructor and destructor to create the assignment operator. For the copy and swap idiom to work, the must already be a deep copy constructor and destructor.

```
IntClass& operator=(IntClass n){
    int * temp = n.data;
    n.data = this->data;
    this->data = temp;
    return *this;
}
```

Questions?