FILE IO, EXCEPTION HANDLING, PASSING, AND OUR FIRST OBJECT (STRUCTS)!

CS202: Computer Science II
Sara Davis
UNR Fall 2022

TOPICS

- Makefile review
- File IO
- Command Line Arguments
- Exception Handling
- Pass by Reference
- Our first object (structs)

MAKEFILE REVIEW DEMO

FILESTREAM OBJECTS

- The <fstream> library is used by c++ for file operations.
  - ofstream (output file stream)
  - ifstream (input file stream)

FILESTREAM OBJECTS

ifstream fin;

fin.open(“example.txt”);

- Declare an object for reading from a file
- Use the object to open read file

ofstream fin;

string name = “example.txt”;

fout.open(filename.c_str())

- Declare an object for writing out
- Use the object to open write file

FILESTREAM OBJECTS: DECLARATION AND INITIALIZATION COMBINED

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

ofstream fout(“example.txt”)
```

FILESTREAM OBJECTS: VERIFYING FILE IS OPENED

```
if (fin.is_open()){

}
```

Or

```
If (fout) {

}
```

READ FROM FILE

- After you confirm that the input file is open, the ifstream object and extraction operators can be used to read from the file.
  - Don't need to worry about anything other than the types showing up in the order you say they will in the read file.

```
fin >> var1 >> var2 >> var3;
```

WRITE TO FILE

- After you confirm that the output file is open, the ofstream object and extraction operators can be used to write to the file

```
fout << "output:" << var1 << endl;
```

CLOSING FILES

```
fin.close()

fout.close()
```

C VS C++

- Differences:
  - No Pointers
  - Uses extraction and insertion operators
  - No EOF check
- Similarities:
  - Close file
  - Setup for loops

COMMAND LINE ARGUMENTS: THE SAME IN C AND C++

```
int main(int arc, char const *argv[])

    if (argc == 1){

        cout << argv[0] << endl;

    }

    return 0;

}
```

EXCEPTION HANDLING

- When writing a class, we may want to tell programmers that they're using the code wrong without revealing it to the end user
- New Keywords:
  - throw
  - try
  - catch

THROW

- Used where the error occurs
- You decide what information you want to send (ie, string error message, int value, etc)
- Return whatever information you decided to send

```
throw “oops!”;
```

TRY

- Try is wrapped around functions we think might throw an exception

```
try{

    trickyFunctionCall();

}
```

CATCH

- Catch is wrapped around what you would like to do if you have an error.

```
catch (Exception e){
    resetValues(object);
    trickyFunctionCall();
}
```

THROW EXAMPLE

```
trickyFunctionCall(int someValue){
    if(someValue < 0){
        throw “oops”;
    }
}
```

TRY CATCH

```
try{
    trickyFunctionCall(value);
}
catch(const char* e){
    cout << e << endl;
    trickyFunctionCall(abs(value));
}
```

EXCEPTION HANDLING DEMO

PASSING ARGUMENTS

- C++ allows you to pass arguments to functions in one of 3 ways:
  - Value
    - Makes a copy of the argument’s value to pass.
  - Address
    - Makes a copy of the argument’s address to pass.
  - Reference
    - Argument is sent into the function.

PASSING ARGUMENTS

- Like mail:
  - Pass by value
    - Copy of the letter is mailed and delivered.
    - You can write on the copy but not the original (shallow copy)
  - Pass by address
    - The address of the letter is delievered
    - You can write on the original by going to the address where the original was sent
  - Pass by reference
    - The actual letter is delivered
    - You can write on the letter.

PASSING BY REFERENCE

- Better than pass by address
  - Pass by address allows pointers to be reassigned. Pass by reference cannot
  - A pointer to a class/struct uses -> reference uses a .
  - A pointer has to be dereferenced (*), but a reference can be used directly

EXAMPLE

```
Void pointerSwap( Donkey* a, Donkey* b){
    Donkey C = a*;
    *a = *b;
     *b = c;
{
Void referenceSwap(Donkey& a, Donkey& b){
     Donkey c = a;
     a= b;
     b = c;
}
```

EXAMPLE

```
void foo (const Donkey& a, Donkey& b){

    b.setThing(a.getThing());

}
```

- Pass by reference is better than pass by value because it uses less memory (by not making a copy each time)
- Pass by reference is better than pass by address, but more dangerous
  - Protect using the const keyword

WHAT IS OBJECT ORIENTED PROGRAMMING (OOP)?

- Organize your program design around data/objects, rather than functions and logic.
  - Example: write a program to read student data from a file and display it.
    - CS 135 (procedural) - write function for read and print, print using loop.
    - CS 202 (OOP) - write a class called student with associated **properties** (variables) [name, GPA, grad year] and **methods** (functions) [read, write].

OOP EXAMPLE

- Design a program that tracks car dealership inventory
  - Need a car class
    - Make
    - Model
    - Year
    - Color
  - Driver class
    - Accesses number of cars of each type by parameter

WHY OOP?

- Procedural programming = function + data
  - Functions can operate on any data-> might change data independently multiple times
  - Data does not belong to any part of the program
    - Code that you’re looking for could be anywhere
  - Functions call functions, calling function may not know called function changed data

WHAT IS A STRUCT?

- Another variable that contains properties and sometimes methods
- Defaults to public
- Typically used when we don’t care about using private or protected (though we can still use access specifiers in a struct)

STRUCT BODY

```
struct Cat{

    int legs;

    bool tail;
};
```

SETTING STRUCT VALUES

```
int main(){
    struct Cat cat;
    cat.legs = 4;
    cat.tail = true;
    cout << "The animal has "<< cat.legs << "legs, and does ";
    if (cat.tail == true){
        cout << "have a tail." << endl;
    else{
        cout << "not have a tail." << endl;
    return 0;
}
```

STRUCT DEMO

YOUR TURN!

- Write a struct for date.
  - It should be purely public
  - Add a print function for today’s date
  - Write the main.cpp code that would set the struct properties

NEXT CLASS

- Strings and arrays of objects
- A brief intro to the 4 pillars of OOP (Encapsulation and Abstraction especially)
- Classes
- Class Constructors/Destructors