EECS 162
Intro to Programming II

Scope

Scope

- Where an identifier is visible or usable in a program
- C++ uses braces, {}, to delimit the bocks
- An identifier usable within the block where it is defined , i.e. in scope
- When the flow of control leaves that block any identifiers declared in the block are also gone

Global Scope

- An identifier declared outside any block of code
  - Makes it available to all blocks in the file!
- Global can be useful, and not dangerous
  - Typically used for constants
- Why?
  - They cannot be changed
  - Often needed in many parts of the program
  - Creates difficulty in the design if they can't be changed in just one place

Global is Bad

- Ideally, a variable is controlled by the authorized programmers who create it
- Global variables can be used and changed by anyone
  - What if 2 coders alternately change the same variable, without knowledge of the other?
  - Makes debugging by either difficult as they may not know the latest value saved there

Local Scope

- Variables declared inside blocks
- With source code control prevents others from changing the values stored
- Allows use of the same identifier across code
  - Consider *i* as a loop control variable
- Variables deleted when block exits

Considerations

```
int x;  //Global so visible anywhere in file
main(){
int z;   // local to main but not visible in any other functions


…..
}
foo(){
  for (int i=0;ctr<10;ctr++  //visible ONLY in loop
  {
          sum+=i;
  }
}
```

Nested Scope

- Blocks can be placed entirely within other blocks
- Scope rule still applies
  - Each variable is visible in the block where it is declared
- An identifier (name) can be used in multiple blocks
  - Each is unique
- Try to avoid as it can be confusing

How Confusing?

```
int i;
main(){
int i;

…..
}
foo(){
  for (int i=0;ctr<10;ctr++
  {
        sum+=i; //which i was intended?
  }
}
```

Supports Decomposition

- For design we need to know what a function does and don't care about how, yet
- Apply the black box concept
  - Each block in your decomposition does something
  - But you don't know how (next level of decomposition)
- Box internals should be isolated
  - Local scope enforces that

Supports Incremental Development

- Incremental Programming
  - Stepwise refinement of program
- Changes should be limited to current block
  - May be a nested block
  - Local variables enforce that
- Global variables would allow changes to propagate to ALL parts of the program!

Scope in OOAD

- Same concerns, just a larger canvas
- Class must encapsulate data members
- Done with private visibility
- Only functions in the class can access them
  - To read or change them
- Protected visibility can extend this access to subclasses

(Lack of) Scope in OOAD

- By design, encapsulation should enforce scope
- Bad programming can change that, unfortunately
- How?
  - Change visibility to public
  - Data member can be changed by in any object
- I've seen this encouraged to make programs more efficient
- How? Avoids function call overhead of accessor and mutator functions
- Cost? Destroys protection of encapsulation!!

Say NO to Global

- Global variables are neither a syntax or logic error
- They are a software engineering error as they violate structured programming principles
- They are a style guide error for 162
  - They are a potential hazard
  - Duplicate identifiers in different scope are confusing
  - They impair readability and clarity

Say NO to Global

- Public data members will demonstrate a lack of understanding of OOP principles
  - A major error, i.e. a major deduction

- Some purists maintain that even protected visibility violates encapsulation
- For our purposes use private or protected, whichever you feel makes your code easier to understand