More Flow of Control Part Deux

CS 211

Adam Sweeney

September 8, 2017

Wichita State University, EECS

More Flow of Control Part Deux
CS 211

Adam Sweeney
September 8, 2017
Wichita State University, EECS

Introduction

- Do Homework 1
- Finish off Chapter 3
- Continue Example for Homework 2

Introduction
- Do Homework 1
- Finish off Chapter 3
- Continue Example for Homework 2

Agenda

- Introduction to Scope
- Some More Loop Information
- Designing a Loop
- Debugging a Loop

Agenda
- Introduction to Scope
- Some More Loop Information
- Designing a Loop
- Debugging a Loop

Introduction to Scope

Introduction to Scope

- We've discussed compound statements
- Compound statements are identified by and contained within
- Our main function is a compound statement
- Our loop bodies that use contain compound statements
- We may want to declare a variable within one of these compound statements

Introduction to Scope
- We've discussed compound statements
- Compound statements are identified by and contained within
- Our main function is a compound statement
- Our loop bodies that use contain compound statements
- We may want to declare a variable within one of these compound statements

Introduction to Scope (cont.)

- Any variable declared within a compound statement, or block, exists solely within that block
- Variables declared in an outer block exist in any nested blocks

Introduction to Scope (cont.)
- Any variable declared within a compound statement, or block, exists solely within that block
- Variables declared in an outer block exist in any nested blocks

Introduction to Scope (cont.)

```
int main(void)
{
    int count = 2;

    for (int i = 1; i <= 3; i++) {
        int count = 4 + i;
        cout << count << endl;
    }

    cout << count << endl;

    return 0;
}
```


Introduction to Scope (cont.)

```
int main(void)
{
    int count = 2;

    for (int i = 1; i <= 3; i++) {
        int count = 4 + i;
        cout << count << endl;
    }

    cout << count << endl;

    return 0;
}
```

1. How many different variables do I have? [A: 3]
2. count is declared twice
3. The loop variable i is local to the for loop, i.e., it does not exist outside of the loop body
4. A second variable called count also exists only within the for loop block, body, compound statement
5. What should I expect to see printed to the screen?

Introduction to Scope (cont.)

The output is the following:

5
6
7
2

Introduction to Scope (cont.)
The output is the following:
5
6
7
2

1. Because I declared a variable inside the for loop, it only exists there
2. Even though it has the same name as another variable, the more local variable of that name is what is used

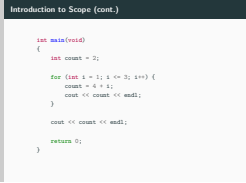
Introduction to Scope (cont.)

```
int main(void)
{
    int count = 2;

    for (int i = 1; i <= 3; i++) {
        count = 4 + i;
        cout << count << endl;
    }

    cout << count << endl;

    return 0;
}
```

1. What's different about this code now? [A: I have one less variable than last time]
2. How does it affect my output?

Introduction to Scope (cont.)

The output is the following:

5
6
7
7

Introduction to Scope (cont.)
The output is the following:
5
6
7
7

1. Because the variable count is not declared inside the for loop, it is using what was declared at the beginning of the main function

Introduction to Scope (cont.)

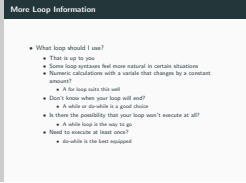
- Scope can be our friend, or our enemy
- Now that we are more aware of it, we can utilize it to our benefit

Introduction to Scope (cont.)
- Scope can be our friend, or our enemy
- Now that we are more aware of it, we can utilize it to our benefit

Some More Loop Information

More Loop Information

- What loop should I use?
  - That is up to you
  - Some loop syntaxes feel more natural in certain situations
  - Numeric calculations with a variale that changes by a constant amount?
    - A for loop suits this well
  - Don't know when your loop will end?
    - A while or do-while is a good choice
  - Is there the possibility that your loop won't execute at all?
    - A while loop is the way to go
  - Need to execute at least once?
    - do-while is the best equipped



1. At the end of the day, the choice of loop is up to you

More Loop Information (cont.)

- Loop variables need to be initialized
- "Just avoid infinite loops"
- The `break` statement is not exclusive to `switch`
  - It can be used in any loop, if there is a need to stop the iterations prematurely
  - Do not rely on `break` for normal exits
  - Design your loops properly, `break` is an exception, not the rule

More Loop Information (cont.)
- Loop variables need to be initialized
- "Just avoid infinite loops"
- The break statement is not exclusive to switch
  - It can be used in any loop, if there is a need to stop the iterations prematurely
  - Do not rely on break for normal exits
  - Design your loops properly, break is an exception, not the rule

1. We can avoid infinite loops by examining our loop starting conditions, our Boolean expression, and our update action

Designing a Loop

Designing a Loop

Designing a Loop

- There are three things we need to design
  1. The body of the loop
  2. The initializing statements
  3. The end conditions
- How should we go about designing these parts?

└Designing a Loop

Designing a Loop

- There are three things we need to design
  1. The body of the loop
  2. The initializing statements
  3. The end conditions
- How should we go about designing these parts?

Designing a Loop (cont.)

- Say what you want to do
- Take what you said and create pseudo-code
- Let's try some examples

Designing a Loop (cont.)
- Say what you want to do
- Take what you said and create pseudo-code
- Let's try some examples

1. Many times, simply saying it to yourself, even out loud, can give you a clue as to what your code should look like
2. pseudocode is not formal code, but sentence fragments written/typed that resembles code in form and behavior
3. pseudocode is effective for designing your whole program, not just a loop

Designing a Loop (cont.)

- Say I want to take integer input from the user a certain number of times, and return the sum of the numbers entered
- Assume we know how many times we will prompt the user, and have stored this information in the variable `thisMany`

Designing a Loop (cont.)
- Say I want to take integer input from the user a certain number of times, and return the sum of the numbers entered
- Assume we know how many times we will prompt the user, and have stored this information in the variable thisMany

Designing a Loop (cont.)

- Some possible ways we might say it to ourselves
  - "I need to get `thisMany` numbers and add them up"
  - "For each number I read, up to `thisMany`, I need to keep track of a total sum"
- Both imply repeated tasks, so we know a loop is needed
- The second way of saying it to ourselves takes it a step further
  - Implies the loop style that might suit us best

Designing a Loop (cont.)
- Some possible ways we might say it to ourselves
  - "I need to get thisMany numbers and add them up"
  - "For each number I read, up to thisMany, I need to keep track of a total sum"
- Both imply repeated tasks, so we know a loop is needed
- The second way of saying it to ourselves takes it a step further
  - Implies the loop style that might suit us best

Designing a Loop (cont.)

- Now, let's consider the first of the three things we need to design
- The body of the loop
  - What is it exactly that our loop needs to do?
  - Read an amount of numbers from the user, then add them to a running total sum
- What does pseudocode look like?

Designing a Loop (cont.)
- Now, let's consider the first of the three things we need to design
- The body of the loop
  - What is it exactly that our loop needs to do?
  - Read an amount of numbers from the user, then add them to a running total sum
- What does pseudocode look like?

Designing a Loop (cont.)

Our pseudocode (in progress)

```
repeat thisMany times
    get number from user
    add to total sum
```


Designing a Loop (cont.)
Our pseudocode (in progress)
repeat thisMany times
get number from user
add to total sum

1. We will be adding to this pseudocode, and the using it to write our actual code

Designing a Loop (cont.)

- Let's move on to the second thing we should design, the initializing statements
- What needs to be initialized?
- What should they be initialized to?

Designing a Loop (cont.)
- Let's move on to the second thing we should design, the initializing statements
- What needs to be initialized?
- What should they be initialized to?

1. A: The sum variable, and our loop counter

Designing a Loop (cont.)

Our pseudocode (in progress)

```
sum = 0
repeat from i = 1 to thisMany times
    get number from user
    add to total sum
```


Designing a Loop (cont.)

Our pseudocode (in progress)

sum = 0
repeat from i = 1 to thisMany times
get number from user
add to total sum

Designing a Loop (cont.)

- Finally, our end conditions
- What is our end condition?
- How do we get there?

Designing a Loop (cont.)
- Finally, our end conditions
- What is our end condition?
- How do we get there?

1. Our end condition taking `thisMany` inputs
2. We get there by incrementing our counter

Designing a Loop (cont.)

Our final pseudocode

```
sum = 0
repeat from i = 1 to thisMany, step i by +1
    get number from user
    add to total sum
```


Designing a Loop (cont.)
Our final pseudocode
sum = 0
repeat from i = 1 to thisMany, step i by +1
get number from user
add to total sum

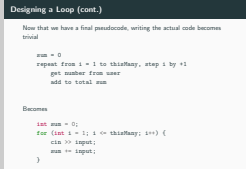
Designing a Loop (cont.)

Now that we have a final pseudocode, writing the actual code becomes trivial

```
sum = 0
repeat from i = 1 to thisMany, step i by +1
    get number from user
    add to total sum
```

Becomes

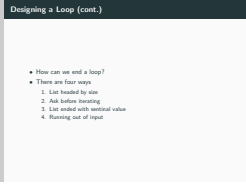
```
int sum = 0;
for (int i = 1; i <= thisMany; i++) {
    cin >> input;
    sum += input;
}
```

1. Note how minimal the changes were to our pseudocode in order to get working code
2. Note also how we did the bulk of the work before ever trying to write any code
3. When we have a blueprint to follow, our code just comes right out
4. Naturally, we can and should do this in chunks, especially as our programs get larger
5. What changes would we have to make if we wanted a total product instead of a sum?

Designing a Loop (cont.)

- How can we end a loop?
- There are four ways
  1. List headed by size
  2. Ask before iterating
  3. List ended with sentinal value
  4. Running out of input



Designing a Loop (cont.)

- List headed by size
  - If we can know ahead of time how many iterations there are, we can simply loop 'n' times
- Ask before iterating
  - As it states, we can get feedback from the user on whethe we should iterate again
- List ended with sentinal value
  - Ending a list with a value that could never appear under normal circumstances
  - A list of names terminated with the integer -1
  - -1 is the sentinel value
- Running out of input
  - Most common when reading from a while
  - Keep reading until there's nothing left

Designing a Loop (cont.)
- List headed by size
  - If we can know ahead of time how many iterations there are, we can simply loop 'n' times
- Ask before iterating
  - As it states, we can get feedback from the user on whethe we should iterate again
- List ended with sentinal value
  - Ending a list with a value that could never appear under normal circumstances
  - A list of names terminated with the integer -1
  - -1 is the sentinel value
- Running out of input
  - Most common when reading from a while
  - Keep reading until there's nothing left

1. Generally, the method to use is intuitive
2. A wrong method will feel wrong when running the program
3. For example prompting the user at each iteration when they have a large list to enter

Debugging a Loop

Debugging a Loop

- Mistakes happen, even to pros
- A big difference that separates pros from "others" is how they handle the mistakes
- So, here are some tips for debugging loops

Debugging a Loop

- Mistakes happen, even to pros
- A big difference that separates pros from "others" is how they handle the mistakes
- So, here are some tips for debugging loops

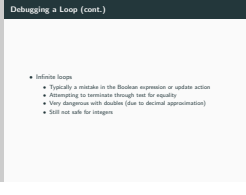
Debugging a Loop (cont.)

- Off-by-one errors
  - Many errors occur on the first or last iteration
  - Loop iterates one too many or one too few times
  - Check your endpoint. Should you be using $<$ or $<=$ (same goes for $>$)
  - If a loop can zero times, does it handle that appropriately?

Debugging a Loop (cont.)
- Off-by-one errors
  - Many errors occur on the first or last iteration
  - Loop iterates one too many or one too few times
  - Check your endpoint. Should you be using < or <= (same goes for >)
  - If a loop can zero times, does it handle that appropriately?

Debugging a Loop (cont.)

- Infinite loops
  - Typically a mistake in the Boolean expression or update action
  - Attempting to terminate through test for equality
  - Very dangerous with doubles (due to decimal approximation)
  - Still not safe for integers



1. Saw a few infinite loops in lab
2. Still not safe for ints just because there is only one way to satisfy equality

Debugging a Loop (cont.)

- Make sure the problem is actually in the loop
- Trace your variables
- Always Be Testing
  - This not throwing code to see what sticks
  - This is controlled testing, with constant feedback to guide us

Debugging a Loop (cont.)
- Make sure the problem is actually in the loop
- Trace your variables
- Always Be Testing
  - This not throwing code to see what sticks
  - This is controlled testing, with constant feedback to guide us

1. Tracing means watch the values change across a few iterations
2. Use cout, or a debugger if you so choose