CSC180: Lecture 8

Wael Aboulsaadat

wael@cs.toronto.edu
http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

General Methods To Control Loops

- Three general methods to control any loop
  - Count controlled loops
  - Ask before iterating
  - Exit on flag condition

Count Controlled Loops

- Count controlled loops are loops that determine the number of iterations before the loop begins
  - The list headed by size is an example of a count controlled loop for input

Exit on Flag Condition

- Loops can be ended when a particular flag condition exists
  - A variable that changes value to indicate that some event has taken place is a flag
  - Examples of exit on a flag condition for input
    - List ended with a special value

Exit on Flag Pitfall

- Consider this loop to identify a student with a grade of 90 or better

```
int n = 1;
grade = compute_grade(n);
while (grade < 90)
{
    n  = n +1;
    grade = compute_grade(n);
}

printf( "Student number %d  has a score %d “, n, grade );
```

The Problem

- The loop on the previous slide might not stop at the end of the list of students if no student has a grade of 90 or higher
  - It is a good idea to use a second flag to ensure that there are still students to consider
  - The code on the following slide shows a better solution

The Exit On Flag Solution

- This code solves the problem of having no student grade at 90 or higher

```
int n=1;
grade = compute_grade(n);
while (( grade < 90) && ( n <= number_of_students))
{
             n = n + 1;
            grade = compute_grade(n);
}

if (grade > 90)
    printf( "Student number %d  has a score %d “, n, grade );
else
    printf(“No student has a high score.“ );
```

Nested Loops

- The body of a loop may contain any kind of statement, including another loop
  - When loops are nested, all iterations of the inner loop are executed for each iteration of the outer loop
  - Give serious consideration to making the inner loop a function call to make it easier to read your program

Example

- Print the factorial of all numbers between 1 and 100

Example

- Print the factorial of all numbers between 1 and 100

Outside loop ➔ iterate on numbers from 1 to 100

Inside loop ➔ for each number calc factorial

Loop pitfalls

- Pitfalls involving loops include
  - Off-by-one errors in which the loop executes one too many or one too few times
  - Infinite loops usually result from a mistake in the Boolean expression that controls the loop

Fixing Off By One Errors

- Check your comparison:

  should it be < or <=?

- Check that the initialization uses the correct value

  0 and <

  1 and <=

- Does the loop handle the zero iterations case?

Fixing Infinite Loops

- Check the direction of inequalities:
  < or > ?

- Test for < or > rather than equality (= =)
  - E.g. doubles are really only approximations

```
double Num1, Num2;
while (Num1 < Num2)
vs.
while (Num1 == Num2)
```

More
Loop Debugging Tips

- Be sure that the mistake is really in the loop
- Trace the variable to observe how the variable changes
  - Tracing a variable is watching its value change during execution
    - Many systems include utilities to help with this
  - printf statements can be used to trace a value on linux/gcc

Debugging Example

- The following code is supposed to conclude with the variable product containing the product of the numbers 2 through 5

```
int next = 2,
    product = 1;
while (next < 5)
{
        next = next + 1;
        product = product * next;
}
```

Tracing Variables

- Add temporary printf statements to trace variables

```
int next = 2,
    product = 1;
while (next < 5)
{
    next      = next + 1;
    printf( "next = %d “, next );
    product = product * next;
    printf( "product= %d“, product );
}
```

Loop Testing Guidelines

- Every time a program is changed, it must be retested
  - Changing one part may require a change to another
- Every loop should at least be tested using input to cause:
  - Zero iterations of the loop body
  - One iteration of the loop body
  - One less than the maximum number of iterations
  - The maximum number of iteratons

Loop Testing Guidelines

- Every time a program is changed, it must be retested (Regression testing)
  - Changing one part may require a change to another

- Every loop should at least be tested using input to cause: (Boundary testing)
  - Zero iterations of the loop body
  - One iteration of the loop body
  - One less than the maximum number of iterations
  - The maximum number of iteratons

Starting Over

- Sometimes it is more efficient to throw out a buggy program and start over
  - The new program will be easier to read
  - The new program is less likely to be as buggy
  - You may develop a working program faster than if you repair the bad code
    - The lessons learned in the buggy code will help you design a better program faster

Data types

Data Types and Expressions

- 2 and 2.0 are not the same number
  - A whole number such as 2 is of type int
  - A real number such as 2.0 is of type double

- Numbers of type int are stored as exact values
- Numbers of type double may be stored as approximate values due to limitations on number of significant digits that can be represented

Writing Integer constants

- Type int does not contain decimal points
  - Examples: 34 45 1 89

Writing Double Constants

- Type double can be written in two ways
  - Simple form must include a decimal point
    - Examples: 34.1 23.0034 1.0 89.9

  - Floating Point Notation (Scientific Notation)
    - Examples: 3.41e1 means 34.1
      3.67e17 means 367000000000000000.0
      5.89e-6 means 0.00000589
  - Number left of e does not require a decimal point
  - Exponent cannot contain a decimal point

Other Number Types

- Various number types have different memory requirements
  - More precision requires more bytes of memory
  - Very large numbers require more bytes of memory
  - Very small numbers require more bytes of memory

Number Types

Type Name	Memory Used	Size Range	Precision
short (also called *short int*)	2 bytes	–32,767 to 32,767	(not applicable)
int	4 bytes	–2,147,483,647 to 2,147,483,647	(not applicable)
long (also called *long int*)	4 bytes	–2,147,483,647 to 2,147,483,647	(not applicable)
float	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
double	8 bytes	approximately 10^{-308} to 10^{308}	15 digits
long double	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits

These are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types *float*, *double*, and *long double* are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

Integer types

- long or long int (often 4 bytes)
  - Equivalent forms to declare very large integers

```
long  big_total;
long int big_total;
```

- short or short int (often 2 bytes)
  - Equivalent forms to declare smaller integers

```
short small_total;
short int small_total;
```

Floating point types

- long double (often 10 bytes)
  - Declares floating point numbers with up to 19 significant digits

```
long double big_number;
```

- float (often 4 bytes)
  - Declares floating point numbers with up to 7 significant digits

```
float not_so_big_number;
```

Type char

- Computers process character data too
- char
  - Short for character
  - Can be any single character from the keyboard

- To declare a variable of type char:
-

```
char letter;
```

char literals

- Character literals are enclosed in single quotes

```
char letter = 'a';
```

- Strings of characters, even if only one character is enclosed in double quotes
  - "a" is a string of characters containing one character
  - 'a' is a value of type character

Type Compatibilities

- In general store values in variables of the same type
  - This is a type mismatch:

```
int int_variable;
int_variable = 2.99;
```

  - If your compiler allows this, int_variable will most likely contain the value 2, not 2.99

int ←→ double (part 1)

- Variables of type double should not be assigned to variables of type int

```
int int_variable;
double double_variable;
double_variable = 2.00;
 int_variable = double_variable;
```

  - If allowed, int_variable contains 2, not 2.00

int ←→ double (part 2)

- Integer values can normally be stored in variables of type double

```
double double_variable;
double_variable = 2;
```

  - double_variable will contain 2.0