Lecture 5: Loops

Sarah Nadi
nadi@ualberta.ca
Department of Computing Science
University of Alberta

CMPUT 201 - Practical Programming Methodology

[With material/slides from Guohui Lin, Davood Rafei, and Michael Buro. Most examples taken from K.N. King's book]

UNIVERSITY OF ALBERTA

Agenda

- The `while` statement
- The `do-while` statement
- The `for` statement
- `break, continue, goto`
- The `null` statement

Readings

- Textbook Chapter 6

What is a Loop?

- A statement that repeatedly executes some other statement(s) that form its *loop body*
- A loop has a *controlling expression* that is evaluated each time the loop body is executed
    - if the expression is “true” (non-zero): continue the loop
    - if the expression is “false” (zero): terminate the loop

Iteration Statements

- `while`
- `do`
- `for`

The `While` Statement

```
while (controlling expression){ statement }
```

- controlling expression is tested **before** the loop body is executed

- if controlling expression is nonzero (i.e., true), loop body is executed and controlling expression gets tested again

- if controlling expression is zero (i.e., false), loop body is not executed

The `While` Statement

Cont'd

- unless there is explicit early exiting from the loop, the controlling expression will be false when the loop terminates

  - e.g., when the following loop terminates, we have i >= n:

    ```
    i = 1;
    while (i < n)
      i *=  2;
    ```

- Infinite while-loop

  ```
  while (1) statement
  ```

  - have to use loop-exiting statement to terminate

Infinite Loop Example

```c
/* Converts a Fahrenheit temperature to Celsius */
#include <stdio.h>
#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f / 9.0f)
int main(void) {
    float fahrenheit, celsius;
    while (1) { /* replacing previous for (;;) { */
        printf("Enter Fahrenheit temperature (non-number to
                 quit): ");
        if (scanf("%f", &fahrenheit) == 1) {
            celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
            printf("Celsius equivalent: %.1f\n", celsius);
        }
        else break;
    }
    return 0;
}
```

Infinite Loop Example

```
/* Converts a Fahrenheit temperature to Celsius */
#include <stdio.h>
#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f / 9.0f)
int main(void) {
    float fahrenheit, celsius;
    while (1) { /* replacing previous for (;;) { */
        printf("Enter Fahrenheit temperature (non-number to
                 quit): ");
        if (scanf("%f", &fahrenheit) == 1) {
            celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;
            printf("Celsius equivalent: %.1f\n", celsius);
        }
        else break;
    }
    return 0;
}
```

Loop-exiting statement
(more on break later)

The `do` Statement

```
do { statement } while (controlling expression)
```

- the loop body is executed, THEN controlling expression is evaluated. This means that the loop body is executed at least once.
- Always use { … } to enclose the loop body and always put a ; after the controlling expression.
- unless there is explicit early exiting from the loop, the controlling expression will be false when the loop terminates. e.g., when the following loop terminates, we will have i >= n.

```
i = 1;
do {
  i *= 2;
} while (i < n);
```

The `for` Statement

```
for (expr1; expr2; expr3) { statement }
```

- `expr1` is executed only once as an initialization step
- `expr2` is the controlling expression and is evaluated every iteration. If “true”, execute the loop body, else terminate.
- If the loop body is executed, `expr3` is executed after the loop body.
- A for loop can be re-written as a while loop:

```
expr1;
while (expr2){
   statement
   expr3;
}
```

`for` Statement Idioms

- Counting up from 0 to `n` - 1:

```
for (i = 0; i < n; i++) ...
```

- Counting up from 1 to `n`:

```
for (i = 1; i <= n; i++) ...
```

- Counting down from `n`-1 to 0:

```
for (i = n - 1; i >= 0; i--) ...
```

- Counting down from `n` to 1:

```
for (i = n; i > 0; i--) ...
```

- Take care of:
  - use of < vs. <= (or > vs. >=)
  - off-by-1 errors (remember arrays are 0-indexed)
  - omitting/missing expressions (e.g., `for ( ; ; )` is an infinite loop)

The comma “,” operator

- Can be used to “glue” multiple expressions into one
- `expr1, expr2`
  - `expr1` is first evaluated. Its value is discarded, but it has a side effect.
  - `expr2` is then evaluated and its resulting value is the value of the whole expression
  - E.g., Assume i = 1 and j = 5 then the expression `++i, i + j` evaluates to 7
- The comma operator is useful in places where a single expression is allowed. For example, you may want to have multiple initialization expressions in your for loop:
  - `for (i = 0, j = 0; i < n; i++)`

Group Exercise: Printing a Table of Squares (p102)

```
This program prints a table of squares starting from 3^2.
Enter the maximum number to be squared in the table: 12
         3         9
         4        16
         5        25
         6        36
         7        49
         8        64
         9        81
        10       100
        11       121
        12       144
```

Try with while loop, do-while loop, and for loop

Group Exercise!

Exiting from a Loop using `break`

- Exiting a loop normally happens automatically before or after the loop body, depending on the controlling expression
- To exit in the middle of the loop, we can use the `break` statement
- The break statement jumps out of the innermost loop

```
for (int i = 1; i <= 3; i++){
  for (int j = 1; j <=3; j++){
    if (i != 2)
     printf("%d,%d\n", i, j);
    else
     break;
  }
  printf("End of outer loop\n");
}
```

What is the output of this code?

Exiting from a loop using `goto`

```
identifier: statement
...
...
for ( ; ; ) {
 ...
 goto identifier ;
 ...
}
```

- The goto statement jumps to the statement labeled with “identifier”, which must be in the same function. Assume we want to jump completely to another place in the program:

```
while ( ... ) {
   switch ( ... ) {
      ...
      goto while-loop_done; /* break won’t work here because it
           will only exit the switch statement but not the loop */
      ...
   }
}
while-loop_done: ...
```

Exiting from a loop using `goto`

```
identifier: statement
...
...
for ( ; ; ) {
 ...
 goto identifier ;
 ...
}
```

There’s a famous article by Dijkstra about why goto is considered harmful. Generally speaking, goto statements make the program much harder to read & understand

- The goto statement jumps to the statement labeled with “identifier”, which must be in the same function. Assume we want to jump completely to another place in the program:

```
while ( ... ) {
   switch ( ... ) {
      ...
      goto while-loop_done; /* break won’t work here because it
           will only exit the switch statement but not the loop */
      ...
   }
}
while-loop_done: ...
```

Skipping Iterations Using `continue`

- The continue statement does not completely exit the loop, but only ends the current iteration
- It should only be used inside loops

```
int n = 0;
int sum = 0;
while (n < 10) {
  scanf("%d", &i);
  if ( i == 0 )
    continue;
  sum += i;
  n++;
  /* continue jumps to here (i.e., the end of the loop) */
}
```

The `null` Statement

- The null statement is an empty statement that only contains a semi-colon and does nothing: `;`

- Can be useful if you have a for loop where all the logic is already done in the controlling expression (e.g., checking if n is prime by looking for a divisor d):

```
for (d = 2; d < n && n % d != 0; d++);
```

- Incorrectly placing a semicolon after a loop or if condition will result in a null statement

(Variation of p123, Ex8): Write a program that prints a one- month calendar. The program takes two arguments: number of days in the month and the day of the week on which the month begins.

For example:

```
./calendar 31 3
```

would print

```
           1   2   3   4   5
   6   7   8   9  10  11  12
  13  14  15  16  17  18  19
  20  21  22  23  24  25  26
  27  28  29  30  31
```

Group Exercise!