for Loop Lesson 3 Outline

1. for Loop Lesson 3 Outline
2. for Loop with a float Counter: BAD!
3. float Counter Example #1
4. float Counter Example #2
5. Why float Counters are BAD BAD BAD
6. BAD float Counter Example #1
7. BAD float Counter Example #2
8. Replace float Counter with int #1
9. Replace float Counter with int #2
10. Debugging a for Loop #1
11. Debugging a for Loop #2
12. Debugging a for Loop #3
13. Debugging a for Loop #4
14. Debugging a for Loop #5
15. Changing Loop Bounds Inside Loop #1: BAD
16. Changing Loop Bounds Inside Loop #2: BAD
17. Changing Loop Index Inside Loop #1: BAD
18. Changing Loop Index Inside Loop #2: BAD
for Loop with a float Counter: BAD!

All of the examples of for loops that we’ve seen so far have used int counters.

In principle, C also allows float counters.

But, using a float counter is BAD BAD BAD BAD.
float Counter Example #1

#include <stdio.h>

int main ()
{
    const float initial_sum = 0.0;
    const int program_success_code = 0;
    float real_count;
    float sum;

    sum = initial_sum;
    for (real_count = 1.0;
         real_count <= 10.0; real_count++) {
        sum = sum + real_count;
    } /* for real_count */
    printf("After the loop:\n");
    printf(" real_count = %f, sum = %f\n",
           real_count, sum);
    return program_success_code;
} /* main */
float  Counter Example #2

% gcc -o forreal forreal.c
% forreal

After the loop:
real_count = 11.000000, sum = 55.000000

This is **BAD BAD BAD**. Why?
Why float Counters are BAD BAD BAD BAD

float counters are generally considered to be BAD BAD BAD BAD programming practice, because:

- a float counter is an approximation, and
- therefore a loop with lots of iterations will accumulate a lot of error in the counter variable, as the error from each approximation adds up.
BAD float Counter Example #1

```c
#include <stdio.h>

int main ()
{ /* main */
  const float pi                   = 3.1415926;
  const int    program_success_code = 0;
  float radians;

  for (radians = 0;
       radians <= 100.0 * pi;
       radians = radians + pi / 5.0) {
    printf("radians = %19.15f\n", radians);
  } /* for radians */
  printf("After the loop:\n");
  printf("  100 * pi = %19.15f\n", 100.0 * pi);
  printf("  radians = %19.15f\n", radians);
  return program_success_code;
} /* main */
```
BAD float Counter Example #2

```
gcc -o forreal2 forreal2.c
forreal2
radians   =   0.000000000000000
radians   =   0.628318488597870
radians   =   1.256636977195740
radians   =   1.884955525398254
radians   =   2.513273954391479
...
```

radians = 312.901885986328125
radians = 313.530212402343750
radians = 314.158538818359375

After the loop:

```
100 * pi = 314.159250259399414
radians  = 314.786865234375000
```

This has been a deadly problem in real life. See:

http://www.ma.utexas.edu/users/arbogast/misc/disasters.html
#include <stdio.h>

int main ()
{ /* main */
    const float pi = 3.1415926;
    const int   program_success_code = 0;
    float radians;
    int radians_counter;

    for (radians_counter = 0;
         radians_counter <= 500;
         radians_counter++) {
        radians = radians_counter * pi / 5.0;
        printf("radians = %19.15f\n", radians);
    } /* for radians */
    printf("After the loop:\n");
    printf("  100.0 * pi = %19.15f\n", 100.0 * pi);
    printf("  radians = %19.15f\n", radians);
    printf("  radians_counter = %3d\n", radians_counter);
    return program_success_code;
} /* main */
Replace \texttt{float} Counter with \texttt{int} \#2

\begin{verbatim}
\% gcc -o forreal2int forreal2int.c
\% forreal2int
radians = 0.0000000000000000
radians = 0.628318488597870
radians = 1.256636977195740
radians = 1.884955644607544
radians = 2.513273954391479
...
radians = 312.902618408203125
radians = 313.530944824218750
radians = 314.159240722656250
After the loop:
100.0 * pi = 314.159250259399414
radians = 314.159240722656250
radians_counter = 501
\end{verbatim}

Notice that there’s no \texttt{accumulated} error from approximating \texttt{float} quantities, because each approximation is independent of the others.
Debugging a for Loop #1

Suppose you have a program that has a for loop, and it looks like the for loop has a bug in it.

Assuming that the bug isn’t obvious just from looking, how do we figure out where the bug is?

One thing we can try is to put some printf statements inside the loop body.

Often, the output of the loop body printf statements will tell us where to find the bug.

When we’ve made a change, we can check to make sure that things are going well, using the same printf statements inside the loop body.

Once we know that the loop is debugged, we can delete the printf statements from inside the loop body.
# Debugging a for Loop #2

```c
#include <stdio.h>

int main ()
{ /* main */
    const int initial_sum = 0;
    const int program_success_code = 0;
    int initial_value, final_value, count;
    int sum;

    printf("What are the summation limits?\n");
    scanf("%d %d", &initial_value, &final_value);
    sum = initial_sum;
    for (count = initial_value;
        count <= final_value; count++) {
        sum = sum * count;
    } /* for count */
    printf("The sum from %d to %d is %d.\n",
        initial_value, final_value, sum);
    return program_success_code;
} /* main */
```

```
gcc -o sumbad sumbad.c
sumbad
What are the summation limits?
1 5
The sum from 1 to 5 is 0.
```
Debugging a for Loop #3

#include <stdio.h>

int main ()
{ /* main */
    const int initial_sum = 0;
    const int program_success_code = 0;
    int initial_value, final_value, count;
    int sum;
    printf("What are the summation limits?\n");
    scanf("%d %d", &initial_value, &final_value);
    sum = initial_sum;
    for (count = initial_value;
        count <= final_value; count++) {
        sum = sum * count;
        printf("count = %d, sum = %d\n", count, sum);
    } /* for count */
    printf("The sum from %d to %d is %d.\n", initial_value, final_value, sum);
    return program_success_code;
} /* main */

% gcc -o sumbad sumbad.c
% sumbad
What are the summation limits?
1 5
count = 1, sum = 0
count = 2, sum = 0
count = 3, sum = 0
count = 4, sum = 0
count = 5, sum = 0
The sum from 1 to 5 is 0.
Debugging a for Loop #4

```c
#include <stdio.h>

int main ()
{ /* main */
    const int initial_sum = 0;
    const int program_success_code = 0;
    int initial_value, final_value, count;
    int sum;

    printf("What are the summation limits?\n");
    scanf("%d %d", &initial_value, &final_value);
    sum = initial_sum;
    for (count = initial_value;
         count <= final_value; count++) {
        sum = sum + count;
        printf("count = %d, sum = %d\n", count, sum);
    } /* for count */
    printf("The sum from %d to %d is %d.\n",
           initial_value, final_value, sum);
    return program_success_code;
} /* main */
```

gcc -o sumbad sumbad.c

What are the summation limits?

1 5

count = 1, sum = 1
count = 2, sum = 3
count = 3, sum = 6
count = 4, sum = 10
count = 5, sum = 15
The sum from 1 to 5 is 15.
Debugging a for Loop #5

```c
#include <stdio.h>

int main ()
{ /* main */
    const int initial_sum = 0;
    const int program_success_code = 0;
    int initial_value, final_value, count;
    int sum;

    printf("What are the summation limits?\n");
    scanf("%d %d", &initial_value, &final_value);
    sum = initial_sum;
    for (count = initial_value;
        count <= final_value; count++) {
        sum = sum + count;
    } /* for count */
    printf("The sum from %d to %d is %d.\n", 
        initial_value, final_value, sum);
    return program_success_code;
} /* main */
```

% gcc -o sumbad sumbad.c
% sumbad
What are the summation limits?
1 5
The sum from 1 to 5 is 15.

Changing Loop Bounds Inside Loop #1: BAD

```c
#include <stdio.h>

int main ()
{ /* main */
    const int initial_sum = 0;
    const int program_success_code = 0;
    int initial_value, final_value, maximum_value, count;
    int sum;

    printf("What are the initial, final and ");
    printf("maximum values?\n");
    scanf("%d %d %d", &initial_value, &final_value, &maximum_value);
    sum = initial_sum;
    for (count = initial_value;
        count <= final_value; count++) {
        sum = sum + count;
        if (sum > maximum_value) {
            final_value = final_value - 1;
        } /* if (sum > maximum_value) */
        printf("count = %d, sum = %d, final_value = %d\n", count, sum, final_value);
    } /* for count */
    return program_success_code;
} /* main */
```

BAD!
Changing Loop Bounds Inside Loop #2: BAD

```
gcc -o loopbndschg loopbndschg.c
% loopbndschg
```

What are the initial, final and maximum values?

1 5 3

<table>
<thead>
<tr>
<th>count</th>
<th>sum</th>
<th>final_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
Changing Loop Index Inside Loop #1: BAD

```c
#include <stdio.h>

int main ()
{ /* main */
    const int initial_sum = 0;
    const int program_success_code = 0;
    int initial_value, final_value, maximum_value, count;
    int sum;

    printf("What are the initial, final and ");
    printf("maximum values?\n");
    scanf("%d %d %d",
        &initial_value, &final_value, &maximum_value);
    sum = initial_sum;
    for (count = initial_value;
        count <= final_value; count++) {
        sum = sum + count;
        if (sum > maximum_value) {
            count = count + 1;
        } /* if (sum > maximum_value) */
        printf("count = %d, sum = %d, final_value = %d
", count, sum, final_value);
    } /* for count */
    return program_success_code;
} /* main */
```

BAD!
Changing Loop Index Inside Loop #2: BAD

% gcc -o loopidxchg loopidxchg.c
% loopidxchg

What are the initial, final and maximum values?

1 5 3

count = 1, sum = 1, final_value = 5

count = 2, sum = 3, final_value = 5

count = 4, sum = 6, final_value = 5

count = 6, sum = 11, final_value = 5