Additional Loop types: For loops and do-while Loops

CMSC 104 Section 02
March 25, 2024

Administrative Notes

Quiz 3 next Monday, 3/29

Practice Quiz available later this week; key will be posted on GitHub

Classwork 6 due tonight

Loops

Before Spring Break, we talked about while loops in c

- These are the most general class of loops
  - You can solve all problems with a while loop
- But there are other types of loops that are useful in some situations
  - Can’t always be used to solve a problem
  - But may be simpler and easier than while loops in solving specific problems
- So we introduce them in this class:
  - “for” loops
  - “do-while” loops

Counter-controlled and event-controlled loops

A ***counter-controlled loop*** will run some defined number of times

- The number might be a constant, a variable or the result of some computation
- But when the loop starts it will run a defined number of times

An ***event-controlled loop*** will run until some event occurs

- A variable takes on some value; the user inputs some special value from the keyboard;...
- You never know when the last iteration of the loop will be
- Sometimes called an “indefinite” loop

To reiterate:

You can implement both counter-controlled loops and event-controlled loops using a while loop

- But there are other ways to do it that you need to know about

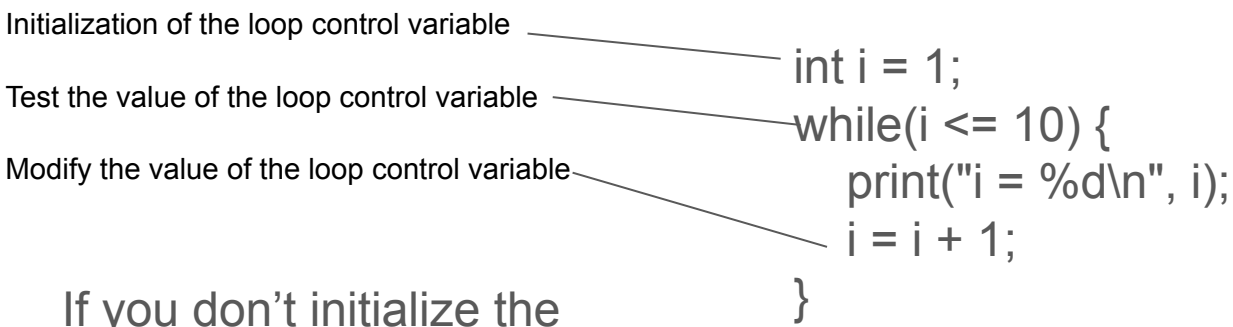
```
int i = 1;
while(i <= 10) {
    print("i = %d\n", i);
    i = i + 1;
}
```

Counter-controlled loop

```
int sum = 0, value;
printf("Enter an integer value: ");
 scanf("%d", &value);
while(value != -1) {
     sum = sum + value;
     printf("Enter another value: ");
     scanf("%d", &value);
}
```

Event-controlled loop

The three parts of a loop control variable:



If you don’t initialize the variable, you’ll get an error because there’s no value

If you don’t test the variable you don’t have a valid loop!

If you don’t modify the value, you’ll get an infinite loop because the test will never fail

“for” loops

Often the simplest way to implement a counter controlled loop

- Because the initialization, test, and modification steps are handled automatically for you

Syntax:

```
int loop_control_variable; //doesn’t have to be an int
for  (loop_control_variable = initial_value; test_condition; modification) {
//loop_body
}
```

“For” loop examples

```
int i;
//counting from zero to 10
for (i = 0; i <= 10; i = i+1) {
   printf("%d\n", i);
}
// we can go backwards too
for (i = 10; i>= 0; i = i -1) {
   printf("%d\n");
}
// we can count by values other than1

for (i = 0; i <= 10; i = i+2) {
   printf("%d\n", i);
}
```

“Do-while” loops

Often used to implement event-controlled loops

Syntax:

```
do {
      //body of loop
} while (condition)
```

The “condition” is always a boolean expression.
The loop will execute again if the condition is true;
the loop will stop if the condition is false

Note: the body of a “do-while” loop must always be executed at least once

- This is not true of a for loop or a while loop

A “do-while” example

```
int num;
do {
   printf("Enter a positive number:  ");
   scanf("%d", @num);
   if (num <= 0) {
       printf("\n That's not positive; try
again. \n");

   }
}  while (num <= 0)
```

```
printf("Enter a positive number:  ");
scanf("%d", @ num);
while (num <= 0) {
   printf("\n That's not positive; try again.\n");
   printf("Enter a positive number:  ");
   scanf("%d", @ num);
}
```

This is an equivalent “while” loop

If you want to be a bad programmer, you can use a “for” loop, too

You can use a for loop for an event-controlled loop, but it’s rather awkward.

```
int num;
printf(“Enter a positive number: “);
scanf (“%d”, &num);
for (: num <=0;) {
     printf("Enter a positive number:  ");
     scanf("%d", @num);
}
```

Don’t do this!!!

What loop to use, when?

Counter-controlled loops:

- It’s almost always easiest to use a “for” loop

Event-controlled loops:

- You can use either a “while” loop or a “do-while” loop
  - Whatever you’re more comfortable with
  - Just remember that a “do-while” loop must always execute at least once

Nesting loops

You can nest one loop inside of another

- Just like you can nest an “if” statement inside another

Print a 2-d table of numbers

```
#include <stdio.h>
int main(){
  int i, j;

  for (i = 0; i <= 10; i = i + 1) {
    for (j = 0; j <= 10; j = j+ 1) {
      printf("%d  ", i * 10 + j);
    }
    printf("\n");
  }

  return 0;
}
```

Print a “checkerboard”

```
#include <stdio.h>
int main(){
  int i, j;

  for (i = 0; i <= 7; i = i + 1) {
    for (j = 0; j <= 7; j = j+ 1) {
      if ((i+j)%2 == 0){
        printf("O");
      } else {
        printf("X");
      } //end of “else”
    }   // end of “for j”
    printf("\n");
  }  // end of “for i”
  return 0;
}
```

More ways to be a “bad” programmer

- “break” statement
  - Lets you break out of a loop early, even if the condition test still returns true
  - Sometimes you just can’t figure out any other way to write a loop that does what you want
  - Don’t do this unless absolutely necessary, and expect problems debugging
- “continue” statement
  - Lets your program run without executing the code in a loop
  - Mostly, useful when you’re developing a piece of a program at a time, and you haven’t gotten the code in that loop working
    - Lets you get the rest of the code working then come back to that loop
  - If you use it at any other time, expect problems debugging your code

“break” example

```c
#include <stdio.h>
int main() {
    int i;
    for(i = 1; i < 10; i++) {
        if (i == 5) {
            break;
        }
        printf("%d ", i);
    }
    printf("Broke out of loop at i = %d\n", i);
    return 0;
}
```

Output:

```
1 2 3 4
Broke out of loop at i = 5
```

“continue” example

```c
#include <stdio.h>
int main() {
    int i;
    for(i = 1; i < 10; i++) {
        if (i == 5) {
            continue;
        }
        printf("%d ", i);
    }
    printf("\nDone\n");
    return 0;
}
```

Output:

```
1 2 3 4 6 7 8 9

Done
```

A nested loop example

Use nested loops to write a prime number calculator. Determine whether each member of a range of numbers is prime by attempting to divide it evenly by each of the smaller numbers.

Strategy:

- Prompt user for upper limit, testing all numbers from 2 to this limit.
- Outer loop: iterate over all numbers from 2 to this limit, testing each in an inner loop to see if it's prime.
    - Inner loop: Iterate over all numbers from 2 to the one less than the number being tested.
    - At the end of an inner loop, if you were never able to evenly divide the number, then it is prime & print it to the user