

Beginning C Programming for Engineers

R. Lindsay Todd

Rensselaer Polytechnic Institute

Lecture 3: Iteration

Outline

- 1 More on Expressions
- 2 More on Assignments
 - Assignment as Operator
 - Short-cut assignments
- 3 Increment and Decrement
 - Postfix operators
 - Prefix operators
- 4 Iteration with `while`
 - Example: Grade average
 - Grade average algorithm
 - Grade average program
- 5 Iteration with `do ... while`
 - Example: Average grades of multiple classes
 - Repeating grade average, algorithm
 - Repeated grade average, program
 - Repeated grade average, execution
- 6 Control variables
 - Computing $n!$

Expressions as Statements

- In C, expressions may be used as statements. Most commonly, this is done with function calls or assignments.
- Examples:

```
1 + 34;  
c - 9;  
a = 5;  
printf("bar = %d\n", bar);
```

Assignment as Operator

- *Assignment* is actually an operator in C. The result of the expression:

```
a = b
```

is `b`, with the *side effect* of assigning the value of `b` to `a`. Assignment takes place after other arithmetic.

- Examples:

```
a = b = c = 1;  
a = b = 2 * (c = 5);
```

Short-cut Assignments

“Short cut” assignment operators combine an operation with an assignment.

<code>a += b</code>	<code>a = a + b</code>
<code>a -= b</code>	<code>a = a - b</code>
<code>a *= b</code>	<code>a = a * b</code>
<code>a /= b</code>	<code>a = a / b</code>
<code>a %= b</code>	<code>a = a % b</code>

For instance, instead of writing:

```
a = a + 1;
```

you could write

```
a += 1;
```

C

Postfix Increment and Decrement

- Postfix increment and decrement operators return the original value of the variable, then increment or decrement the variable.

```
int a, b;
a = b = 10;
printf("%d\n", a++); /* Prints 10 */
printf("%d\n", a); /* Prints 11 */
printf("%d\n", b--); /* Prints 10 */
printf("%d\n", b); /* Prints 9 */
```

C

Prefix Increment and Decrement

- Prefix increment and decrement operators increment or decrement the variable, then return its resulting value.

```
int a, b;
a = b = 10;
printf("%d\n", ++a); /* Prints 11 */
printf("%d\n", a); /* Prints 11 */
printf("%d\n", --b); /* Prints 9 */
printf("%d\n", b); /* Prints 9 */
```

- Remember: If the `++` comes *before* the variable, it increments *before* determining the result.

C

Iteration

- Iteration* is repeating the same statements in a program. Sometimes this is called *looping*. Recall:

```
/* Previous statements */
while (expression) statement
/* Following statements */
```

- The `while` statement executes *statement* as long as *expression* is true. The *expression* is tested before *statement* is ever executed.

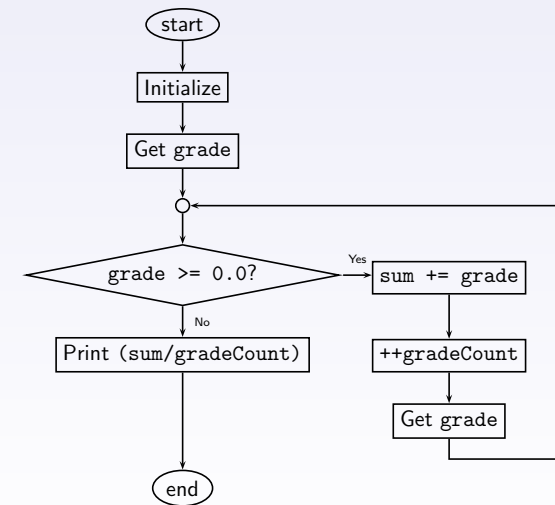
C

Example: Grade average

- We would like a program to average a set of grades.
- Algorithm notes:
 - ▶ We need a *running sum* of grades, and a *running count* of how many grades have been read so far.
 - ▶ We need to read until we get a *sentinel* value — let's use a negative grade to indicate we are done.
 - ▶ Need to be sure we print prompts.

C

Flowchart: Grade Average



C

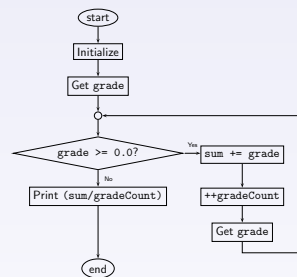
Grade Average and while

Program: sentinel.c

```

#include <stdio.h>

int main()
{
    float grade, sum = 0.0;
    int gradeCount = 0;
    printf("Enter grade: ");
    scanf("%g", &grade);
    while (grade >= 0.0) {
        sum += grade;
        ++gradeCount;
        printf("Enter grade: ");
        scanf("%g", &grade);
    }
    printf("Average: %g\n",
        sum/gradeCount);
    return 0;
}
  
```



Results

```

Enter grade: 10
Enter grade: 90
Enter grade: -1
Average: 50
  
```

C

The do ... while Statement

- Sometimes instead of testing a **while** condition before executing statements, you want to always execute those statements at least once.
- The **do...while** statement can be used.

```

/* Previous statements */
do statement while (expression);
/* Following statements */
  
```

This will execute *statement*. Then, repeatedly test if *expression* is true; if so, loop again.

C

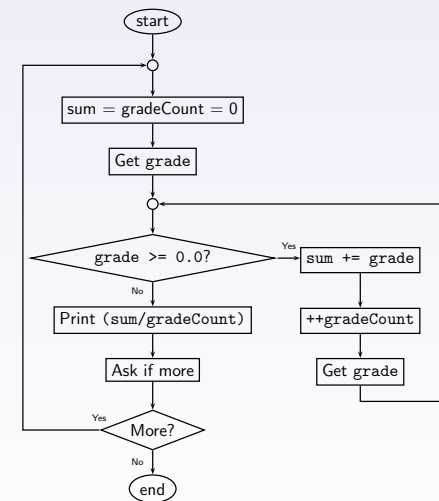
Example: Average grades of multiple classes

We would like a program to compute the average grade of each of a set of classes.

- For each class, operate just like our previous grade average program.
- Need to be careful that the average grade for each class is computed independently of other classes!
- After computing the average for a class, need to see if there is another class.

C

Flowchart: Grade Several Classes



C

Code: Grade Several Classes

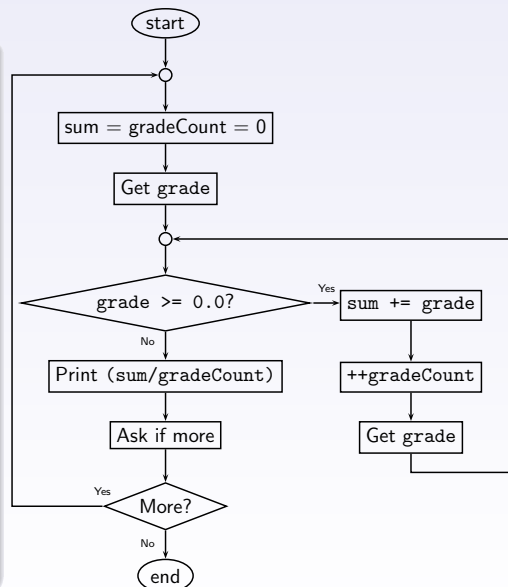
Program: `sentineldw.c`

```

#include <stdio.h>

int main() {
    float grade, sum;
    int gradeCount;
    int another;
    do {
        sum = gradeCount = 0;
        printf("Enter grade: ");
        scanf("%g", &grade);
        while (grade >= 0.0) {
            sum += grade;
            ++gradeCount;
            printf("Enter grade: ");
            scanf("%g", &grade);
        }
        printf("Average: %g\n\n",
            sum/gradeCount);
        printf("Another class: ");
        scanf("%d", &another);
    } while (another != 0);

    return 0;
}
  
```



C

Example: do ... while

Program: `sentineldw.c`

```

#include <stdio.h>

int main() {
    float grade, sum;
    int gradeCount;
    int another;
    do {
        sum = gradeCount = 0;
        printf("Enter grade: ");
        scanf("%g", &grade);
        while (grade >= 0.0) {
            sum += grade;
            ++gradeCount;
            printf("Enter grade: ");
            scanf("%g", &grade);
        }
        printf("Average: %g\n\n",
            sum/gradeCount);
        printf("Another class: ");
        scanf("%d", &another);
    } while (another != 0);

    return 0;
}
  
```

Results

```

Enter grade: 10
Enter grade: 90
Enter grade: -1
Average: 50
  
```

```

Another class: 1
Enter grade: 99
Enter grade: 93
Enter grade: -1
Average: 96
  
```

```

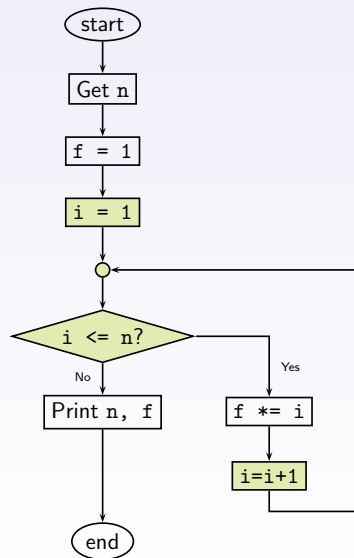
Another class: 0
  
```

C

Computing n!

Problem: compute $n!$ using a loop.

- “Counter” variable, i , ranging from 1 to n .
- Running product f , tracking $i!$.



n! using while

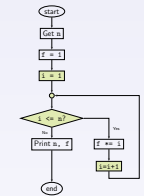
```

/* n! using while. */
#include <stdio.h>

int main() {
    int i, n, f;

    printf("Enter n: ");
    scanf("%d", &n);

    f = 1; /* 0! */
    i = 1;
    while (i <= n) {
        f *= i; /* Now, f = i! */
        ++i;
    }
    printf("%d! = %d\n", n, f);
    return 0;
}
  
```



Results

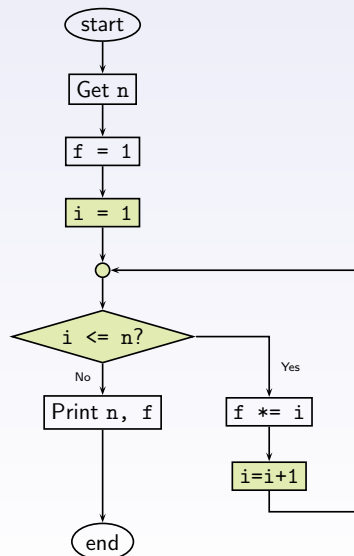
Enter n: 4
4! = 24

Results

Enter n: 0
0! = 1

Control variables

- It is common to have a *control variable* associated with a loop:
 - 1 It is given an *initial value* before the loop.
 - 2 It is tested by a condition at the start of the loop.
 - 3 It is updated to a new value within the body of the loop.
- It is common for the control variable to step from a starting value to an ending value, stepping through intermediate values (like computing $n!$). Often called a *counter* or *index* variable.



The for Loop

- The `for` statement is commonly used when there is a control variable associated with the loop.
- The `for` statement first evaluates *initExpr*. Then, as long as *testExpr* is true, execute *statement*. After each time *statement* is executed, evaluate *stepExpr*.

```

/* Previous statements */
for (initExpr ; testExpr ; stepExpr) statement
/* Following statements */
  
```

- Example:

```

int i;
for (i = 1; i < 10; i += 2) {
    printf("%d\n", i);
}
  
```

n! using for

Program: fact2.c

```

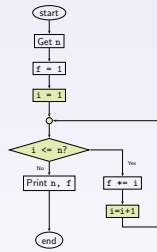
/* n! using for. */
#include <stdio.h>

int
main()
{
    int i, n, f;

    printf("Enter n: ");
    scanf("%d", &n);

    f = 1; /* 0! */
    for (i = 1; i <= n; ++i) {
        f *= i; /* Now, f = i! */
    }
    printf("%d! = %d\n", n, f);
    return 0;
}

```



Results

Enter n: 4
4! = 24

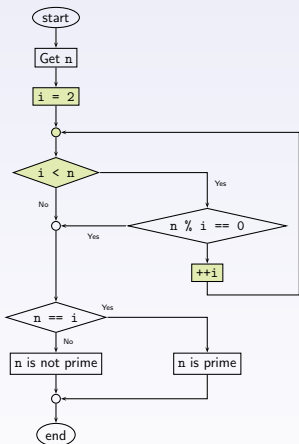
Results

Enter n: 0
0! = 1

The break Statement

- Sometimes we want to end a loop in the middle. Use **break** to immediately end the innermost loop.
- The **break** may occur in any of **while**, **do...while**, or **for**.
- Example: We want a program to determine if a number is *prime*.
 - ▶ Algorithm: Repeatedly try numbers until we get a factor, or run out of numbers.

Prime numbers and break



Results

Enter n: 9997
9997 is not prime (divisible by 13)

Program: prime.c

```

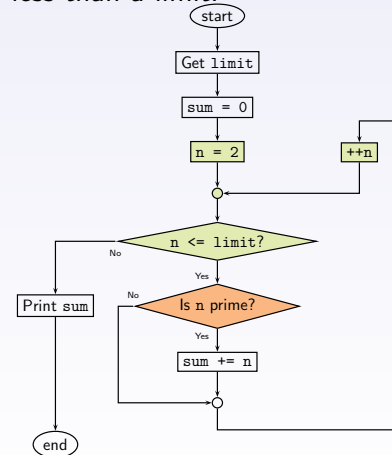
/* Test for prime-ness */
#include <stdio.h>

int
main()
{
    int i, n;
    printf("Enter n: ");
    scanf("%d", &n);
    for (i = 2; i < n; ++i) {
        if (n % i == 0) break;
    }
    if (i == n) {
        printf("%d is prime.\n", n);
    } else {
        printf("%d is not prime (divisible by %d)\n",
            n, i);
    }
    return 0;
}

```

Summing Primes

Problem: Add all prime numbers less than a limit.



Program: primesum.c

```

/* Sum up primes. */
#include <stdio.h>

int
main()
{
    int i, n, limit, sum = 0;

    printf("Enter limit: ");
    scanf("%d", &limit);

    for (n = 2; n <= limit; ++n) {
        /* Test if 'n' is prime. */
        for (i = 2; i < n; ++i) {
            if (n % i == 0) break;
        }
        if (i == n) sum += n;
    }

    printf("Sum of primes <= %d is %d\n",
        limit, sum);
    return 0;
}

```

Nesting Loops

Program: **primesum.c**

```

/* Sum up primes. */
#include <stdio.h>

int
main()
{
    int i, n, limit, sum = 0;

    printf("Enter limit: ");
    scanf("%d", &limit);

    for (n = 2; n <= limit; ++n) {
        /* Test if 'n' is prime. */
        for (i = 2; i < n; ++i) {
            if (n % i == 0) break;
        }
        if (i == n) sum += n;
    }

    printf("Sum of primes <= %d is %d\n",
        limit, sum);
    return 0;
}

```

All loop types can be *nested*.

- The **break** statement only terminates the loop immediately containing it.
- In this example, **break** will only terminate the loop with the control variable **i**.

Results

```

Enter limit: 1000
Sum of primes <= 1000 is 76127

```

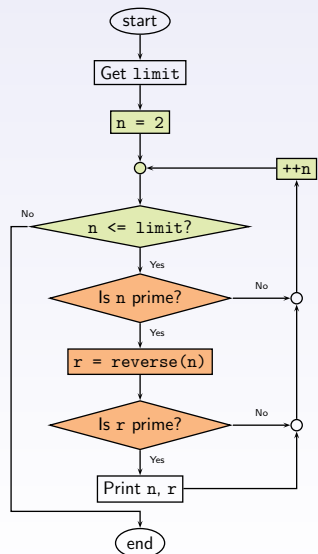
C

The continue Statement

- The **continue** statement restarts a loop.
 - ▶ In a **while** or **do while** loop, the test runs.
 - ▶ In a **for** loop, the increment expression runs, then the test.
- Problem: List prime numbers whose "reverse" are also prime. For example, 13 and its reverse, 31.

C

Primes and Reverses



```

/* Test n, reverse(n) for primeness. */
#include <stdio.h>
int reverse(int n);

int main() {
    int i, limit, n, r;
    printf("Enter limit: ");
    scanf("%d", &limit);
    for (n = 2; n <= limit; ++n) {
        /* Test if 'n' is prime. */
        for (i = 2; i < n; ++i) {
            if (n % i == 0) break;
        }
        if (i != n) continue; /* Not prime */

        r = reverse(n);

        /* Test if 'r' is prime. */
        for (i = 2; i < r; ++i) {
            if (r % i == 0) break;
        }
        if (i != r) continue; /* Not prime */

        printf("%d and %d are prime.\n",
            n, r);
    }
    return 0;
}

```

C

Using continue

```

/* Test n, reverse(n) for primeness. */
#include <stdio.h>
int reverse(int n);

int main() {
    int i, limit, n, r;
    printf("Enter limit: ");
    scanf("%d", &limit);
    for (n = 2; n <= limit; ++n) {
        /* Test if 'n' is prime. */
        for (i = 2; i < n; ++i) {
            if (n % i == 0) break;
        }
        if (i != n) continue; /* Not prime */

        r = reverse(n);

        /* Test if 'r' is prime. */
        for (i = 2; i < r; ++i) {
            if (r % i == 0) break;
        }
        if (i != r) continue; /* Not prime */

        printf("%d and %d are prime.\n",
            n, r);
    }
    return 0;
}

```

Results

```

Enter limit: 100
2 and 2 are prime.
3 and 3 are prime.
5 and 5 are prime.
7 and 7 are prime.
11 and 11 are prime.
13 and 31 are prime.
17 and 71 are prime.
31 and 13 are prime.
37 and 73 are prime.
71 and 17 are prime.
73 and 37 are prime.
79 and 97 are prime.
97 and 79 are prime.

```

C

sentinel.c

```
#include <stdio.h>

int main()
{
    float grade, sum = 0.0;
    int gradeCount = 0;
    printf("Enter grade: ");
    scanf("%g", &grade);
    while (grade >= 0.0) {
        sum += grade;
        ++gradeCount;
        printf("Enter grade: ");
        scanf("%g", &grade);
    }
    printf("Average: %g\n",
        sum/gradeCount);

    return 0;
}
```

sentineldw.c

```
#include <stdio.h>

int main() {
    float grade, sum;
    int gradeCount;
    int another;
    do {
        sum = gradeCount = 0;
        printf("Enter grade: ");
        scanf("%g", &grade);
        while (grade >= 0.0) {
            sum += grade;
            ++gradeCount;
            printf("Enter grade: ");
            scanf("%g", &grade);
        }
        printf("Average: %g\n\n",
            sum/gradeCount);
        printf("Another class: ");
        scanf("%d", &another);
    } while (another != 0);

    return 0;
}
```

fact.c

```
/* n! using while. */
#include <stdio.h>

int main() {
    int i, n, f;

    printf("Enter n: ");
    scanf("%d", &n);

    f = 1; /* 0! */
    i = 1;
    while (i <= n) {
        f *= i; /* Now, f = i! */
        ++i;
    }
    printf("%d! = %d\n", n, f);
    return 0;
}
```

fact2.c

```
/* n! using for. */  
  
#include <stdio.h>  
  
int  
main()  
{  
    int i, n, f;  
  
    printf("Enter n: ");  
    scanf("%d", &n);  
  
    f = 1; /* 0! */  
    for (i = 1; i <= n; ++i) {  
        f *= i; /* Now, f = i! */  
    }  
    printf("%d! = %d\n", n, f);  
    return 0;  
}
```

prime.c

```
/* Test for prime-ness */  
  
#include <stdio.h>  
  
int  
main()  
{  
    int i, n;  
    printf("Enter n: ");  
    scanf("%d", &n);  
    for (i = 2; i < n; ++i) {  
        if (n % i == 0) break;  
    }  
    if (i == n) {  
        printf("%d is prime.\n", n);  
    } else {  
        printf("%d is not prime (divisible by %d)\n",  
            n, i);  
    }  
    return 0;  
}
```

primesum.c

```
/* Sum up primes. */  
  
#include <stdio.h>  
  
int  
main()  
{  
    int i, n, limit, sum = 0;  
  
    printf("Enter limit: ");  
    scanf("%d", &limit);  
  
    for (n = 2; n <= limit; ++n) {  
        /* Test if 'n' is prime. */  
        for (i = 2; i < n; ++i) {  
            if (n % i == 0) break;  
        }  
        if (i == n) sum += n;  
    }  
  
    printf("Sum of primes <= %d is %d\n",  
        limit, sum);  
    return 0;  
}
```

rprime.c

```
/* Test n, reverse(n) for primeness. */
#include <stdio.h>
int reverse(int n);

int main() {
    int i, limit, n, r;
    printf("Enter limit: ");
    scanf("%d", &limit);
    for (n = 2; n <= limit; ++n) {

        /* Test if 'n' is prime. */
        for (i = 2; i < n; ++i) {
            if (n % i == 0) break;
        }
        if (i != n) continue; /* Not prime */

        r = reverse(n);

        /* Test if 'r' is prime. */
        for (i = 2; i < r; ++i) {
            if (r % i == 0) break;
        }
        if (i != r) continue; /* Not prime */

        printf("%d and %d are prime.\n",
            n, r);
    }
    return 0;
}
```