CS1010E Lecture 3

Control Structures: Repetition

Henry Chia (hchia@comp.nus.edu.sg)

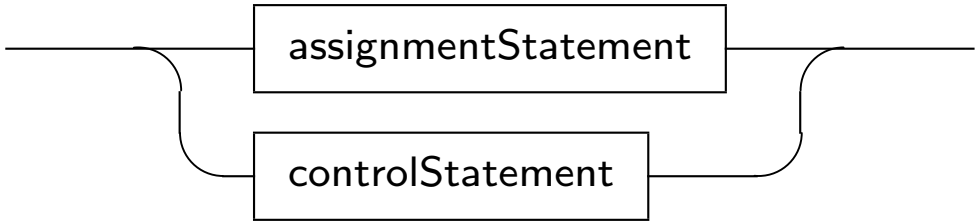
Semester 1 2016 / 2017

Lecture Outline

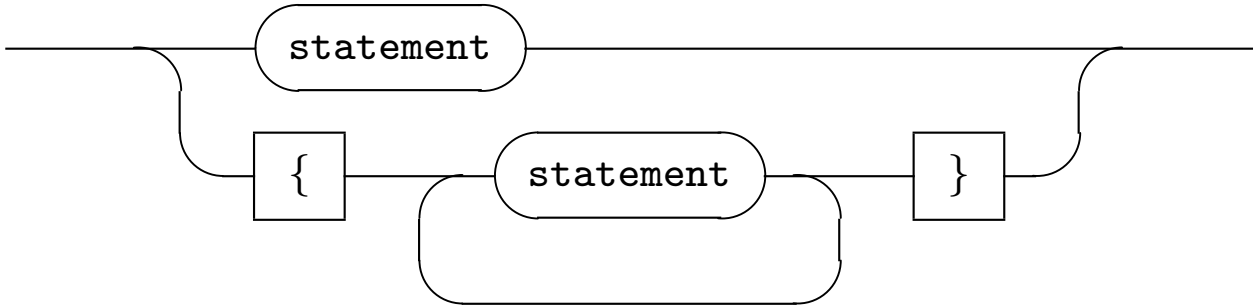
- Repetition control structure
- Multiple user input
- Repetition statements
  - `do..while`
  - `while`
  - `for`
- Types of loops
  - Flag-controlled
  - Result-controlled
  - Count-controlled
- Timeline tracing
- Incremental program development

Statement and Statement Block

statement



stmtBlock



- Control Statements
  - Selection
    - `if..else`
  - **Repetition**
    - `do..while`
    - `while`
    - `for`
- Statement Block – one statement or group of statements

Repetition Statement

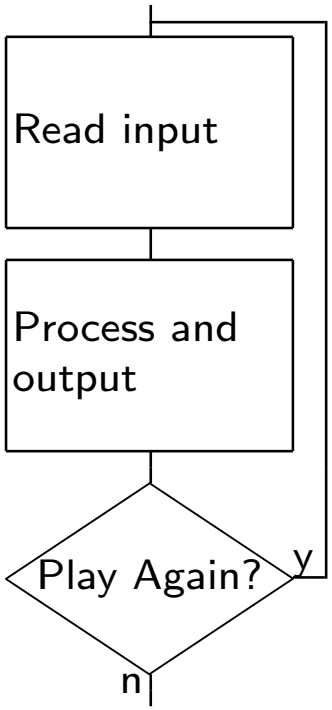
- The repetition statement allows us to repeat (or loop through) a set of steps as long as a condition is true
- Three forms of repetition statement:
  - `do..while` loop
  - `while` loop
  - `for` loop
- Condition must eventually become false to terminate the loop; otherwise infinite (endless) loop

Multiple User Input

- Example: perform miles to kms conversion for a set of input values within a single program run
- How do we indicate that the last input has been read and processed, and the loop should stop?
- Ways to deal with multiple input using:
  - Boolean flag
  - Sentinel value
  - Initial count
- Assess the suitability of different loop constructs

Multiple User Input – "Play Again" Flag

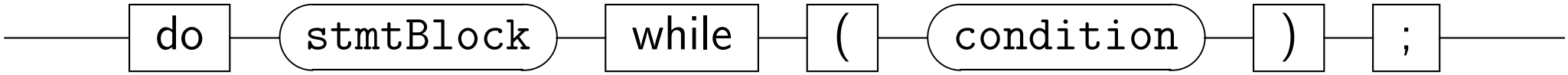
```
Enter distance in miles: 1.0
1.000000 miles = 1.609000 kms
Again? (1=Yes/0=No): 1
Enter distance in miles: 10.0
10.000000 miles = 16.090000 kms
Again? (1=Yes/0=No): 1
Enter distance in miles: 12.3
12.300000 miles = 19.790700 kms
Again? (1=Yes/0=No): 0
```



- Notice that at least one input must be read and processed
- Requires a boolean variable to determine if the loop should continue (or terminate)

Repetition: `do..while` Statement

doWhileStatement

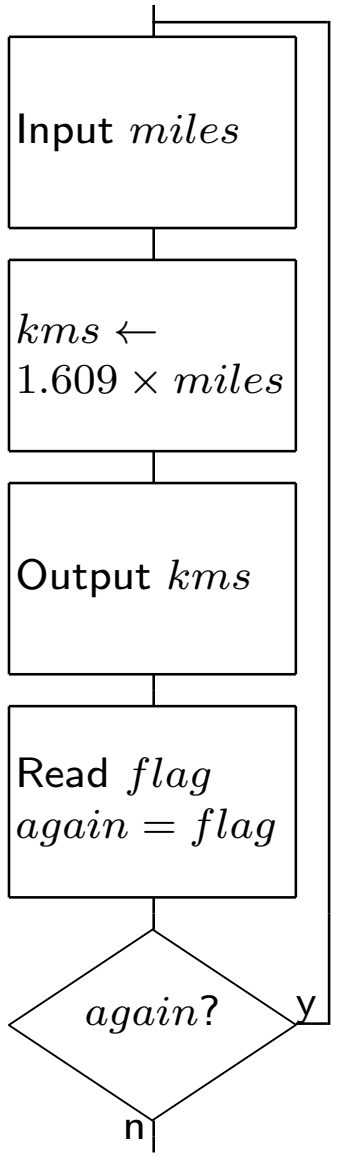



```
do {
   statement(s);
} while (condition); /* note semi-colon */
```

- `do..while` statement block is always executed at least once
- Note the semi-colon at the end of the `do..while` statement
- Curly braces not required for a single-statement block, but encouraged

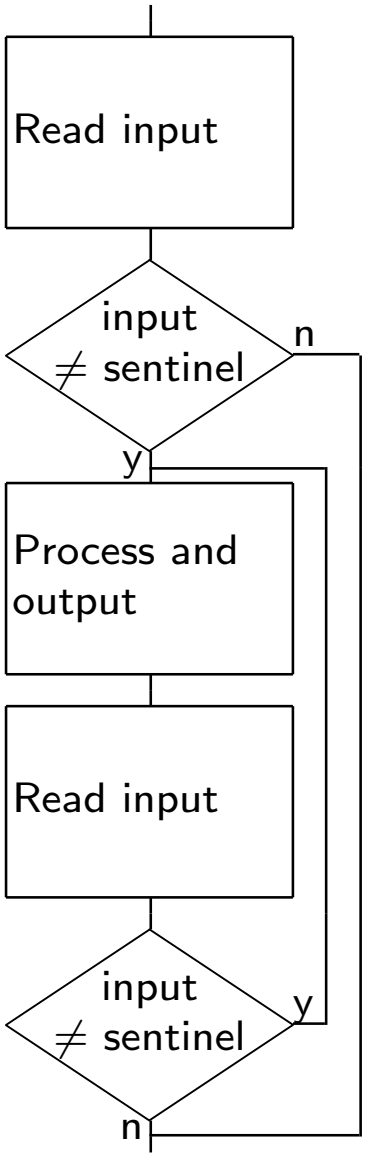
Example: do..while

```c
/*
   This program performs miles to kilometers conversion.
*/
#include <stdio.h>
#include <stdbool.h>

#define KMS_PER_MILE 1.609

int main(void) {
   double miles=0.0, kms=0.0;
   bool again;
   int flag=1;

   do {
      printf("Enter distance in miles: ");
      scanf("%lf", &miles);

      kms = KMS_PER_MILE * miles;

      printf("%f miles = %f kms\n", miles, kms);

      printf("Again? (1=Yes/0=No): ");
      scanf("%d", &flag);
      again = flag;
   } while (again); // again == true

   return 0;
}
```



Multiple User Input – Sentinel Value

```
Enter distance in miles: 1.0
1.000000 miles = 1.609000 kms
Enter distance in miles: 10.0
10.000000 miles = 16.090000 kms
Enter distance in miles: 12.3
12.300000 miles = 19.790700 kms
Enter distance in miles: 0
```



- Suppose only positive values are valid input
- Use a special value (in this case ≤ 0) to indicate the end of user input
- Need to take care of the case where the first input is the sentinel value

Repetition: `while` Statement

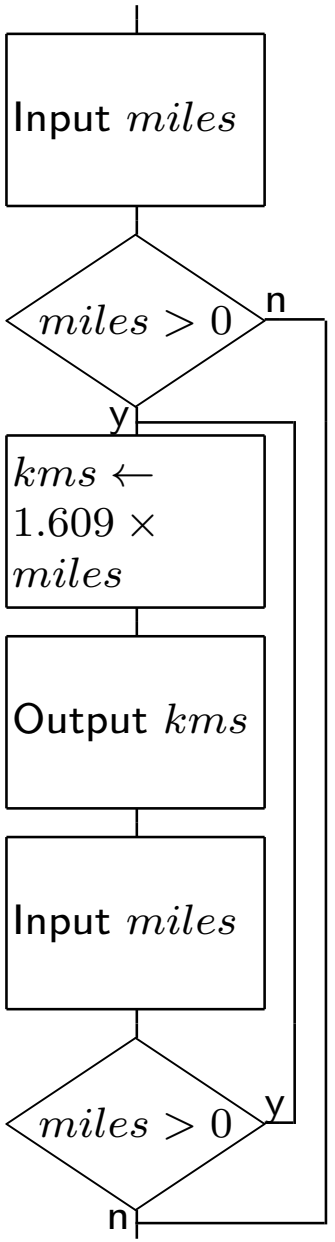
whileStatement


```
while (condition) {
    statements(s);
}
```

- `while` loop is a `do..while` loop with an additional test before the first loop
- Curly braces not required for a single-statement block, but encouraged

Example: `while`

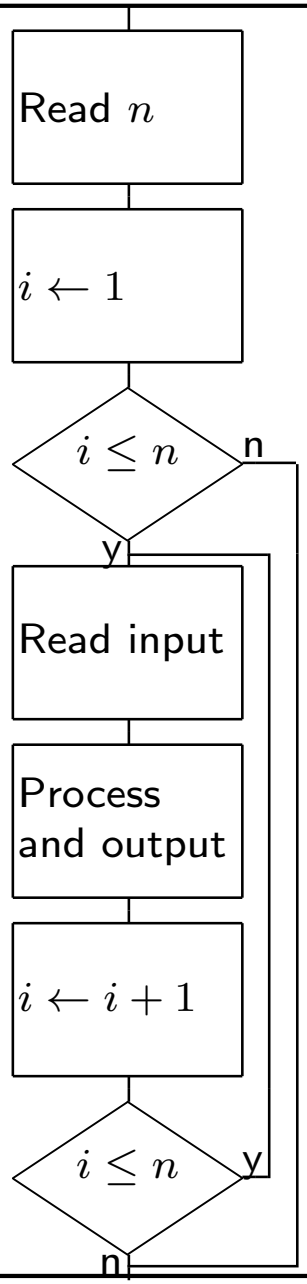
```c
/*
   This program performs miles to kilometers conversion.
*/
#include <stdio.h>

#define KMS_PER_MILE 1.609

int main(void) {
   double miles=0.0, kms=0.0;

   printf("Enter distance in miles: ");
   scanf("%lf", &miles);
   while (miles > 0) {
      kms = KMS_PER_MILE * miles;

      printf("%f miles = %f kms\n", miles, kms);

      printf("Enter distance in miles: ");
      scanf("%lf", &miles);
   }

   return 0;
}
```



Multiple User Input – Initial Count

```
Enter number of input: 3
Enter distance in miles: 1.0
1.000000 miles = 1.609000 kms
Enter distance in miles: 10.0
10.000000 miles = 16.090000 kms
Enter distance in miles: 12.3
12.300000 miles = 19.790700 kms
```

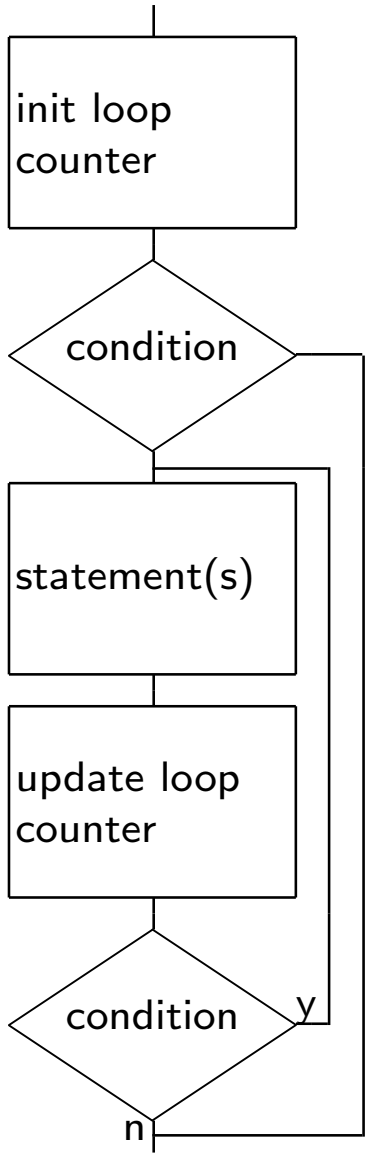
- Read in an initial count, n
- Use a loop counting variable i initialized as 1
- Loop n times to read and process input
- At the end of each loop, increment i by 1
- `while` loop can be used, but `for` loop is preferred



Repetition: `for` Statement

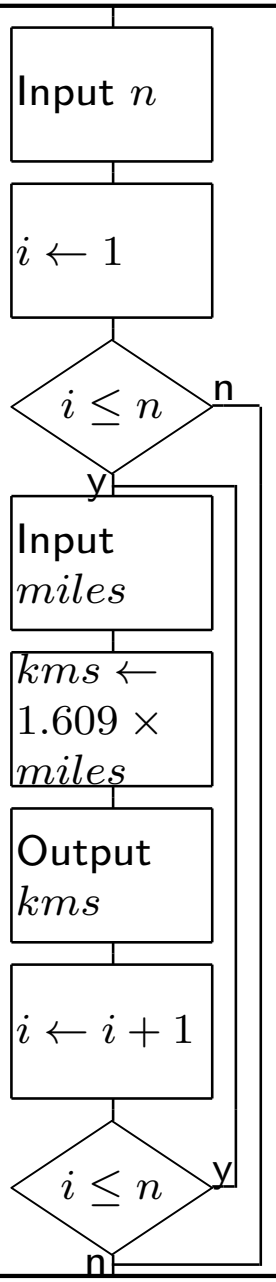
```
for (init; condition; update) {
   statement(s);
}
```

- `init`: initialize the loop counter
- `condition`: check the loop counter
- `update`: modify the loop counter
- Curly braces not required for a single-statement block, but encouraged



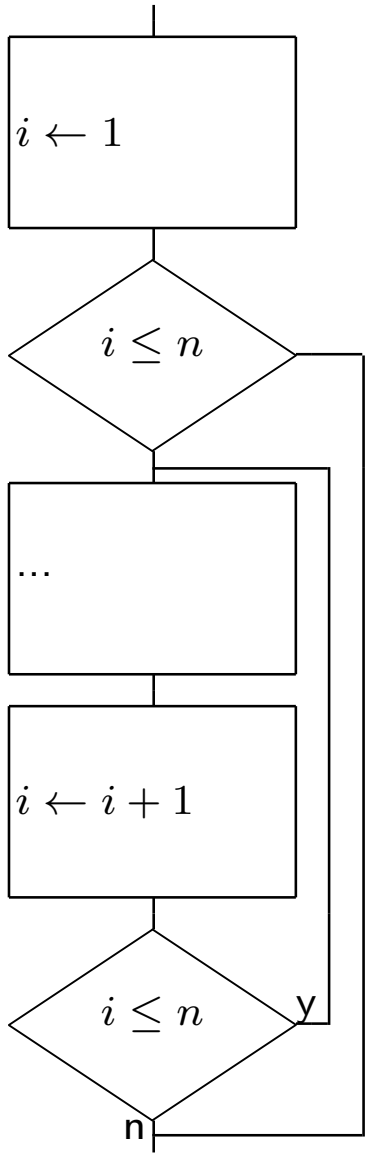
Example: for

```c
/*
   This program performs miles to kilometers conversion.
*/
#include <stdio.h>

#define KMS_PER_MILE 1.609

int main(void) {
   double miles=0.0, kms=0.0;
   int n=0, i=0;

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

   for (i = 1; i <= n; i = i + 1) {
      printf("Enter distance in miles: ");
      scanf("%lf", &miles);

      kms = KMS_PER_MILE * miles;

      printf("%f miles = %f kms\n", miles, kms);
   }

   return 0;
}
```



Count-Controlled Loop

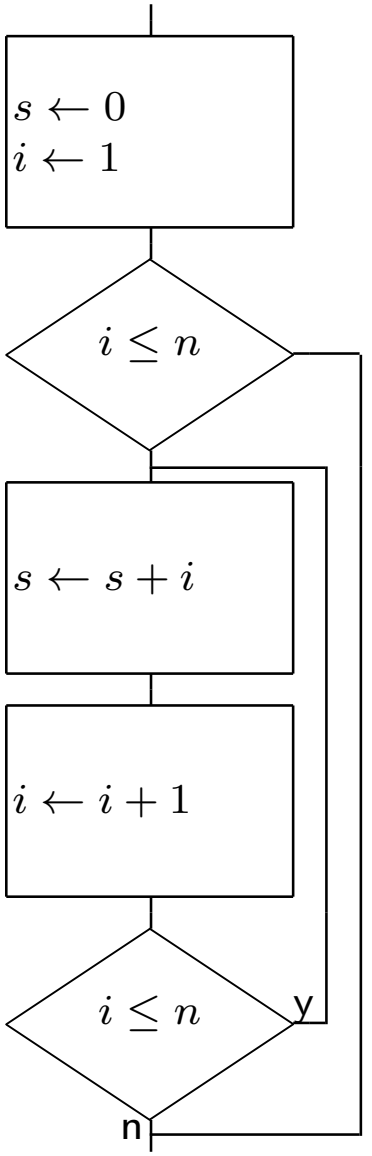
- Count-controlled loop – looping based on a loop counter
- Value of loop counter increments (or decrements) by a consistent amount each time through a loop
- Looping stops when the loop counter reaches a specified value
- Implemented with the `for` loop
- Example: finding $\sum_{i=1}^{n} i$
  - Declare `i` as loop counter
  - Use `i++` as a shorthand for `i = i + 1`



Count-Controlled Loop Using `for`

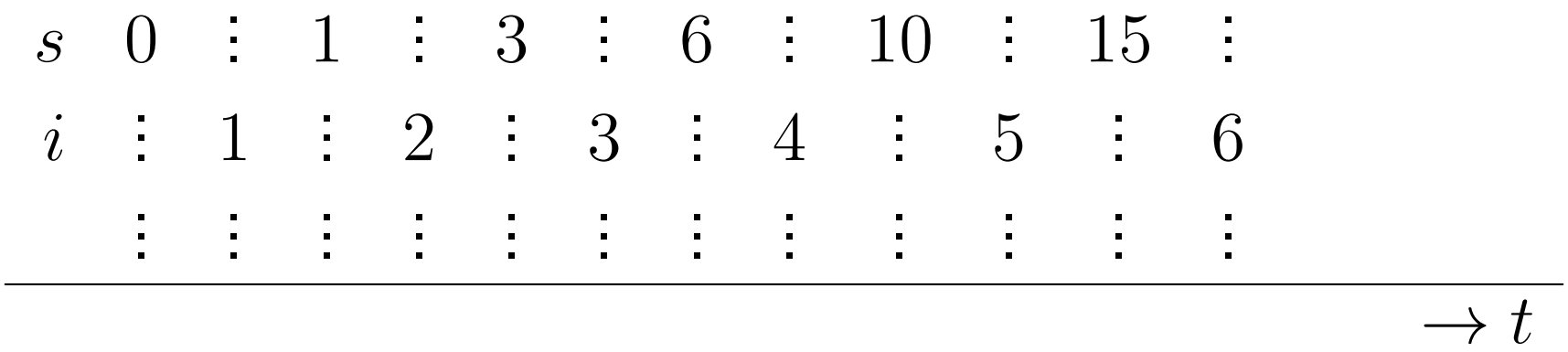
- Given $n \geq 0$, find $s = \sum_{i=1}^{n} i$

```c
/*
   This program computes the sum of
   1 + 2 + ... + n
*/
#include <stdio.h>

int main(void) {
   int n=0, i=0, s=0;

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

   s = 0; /* initialize s to 0 */
   for (i = 1; i <= n; i++) {
      s = s + i;
   }

   printf("Sum is %d\n", s);

   return 0;
}
```



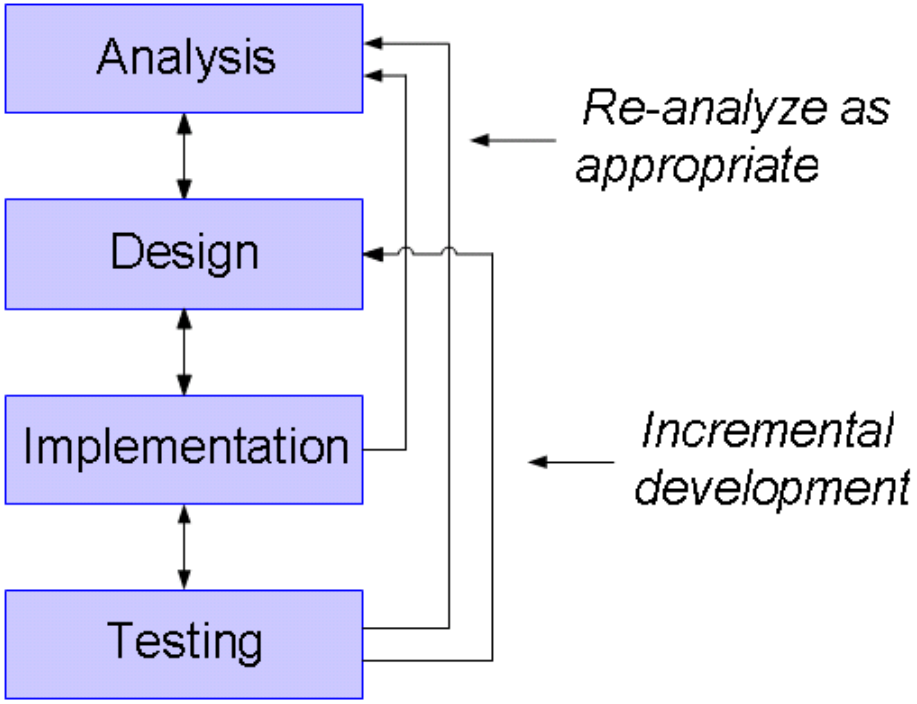
Timeline Tracing

- A timeline trace shows the progress of value changes in variables with respect to time
- Allows one to make inferences on program behavior
- Trace of variables i and s for $n = 5$ of program in preceding slide is shown below:



- To predict the values of s and i as the loop progresses

Incremental Program Development

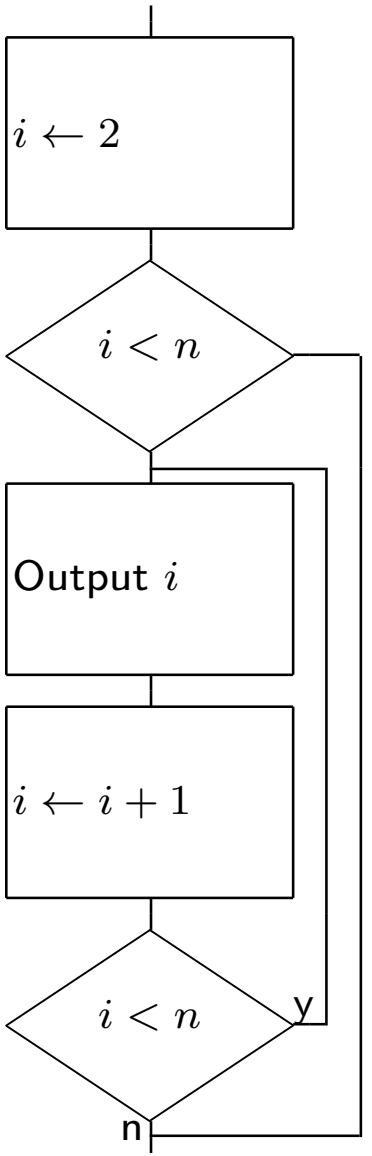


- Example: Given an integer $n(> 1)$, determine if n is prime
  - Loop through all divisors from 2 to $p - 1$
  - If there is a divisor that divides p, p is **not prime**
  - p is prime only when **all** divisors from 2 to $p - 1$ do not divide p

Let's begin...

- Given $n > 1$, output integers $2, 3, \ldots, n-1$

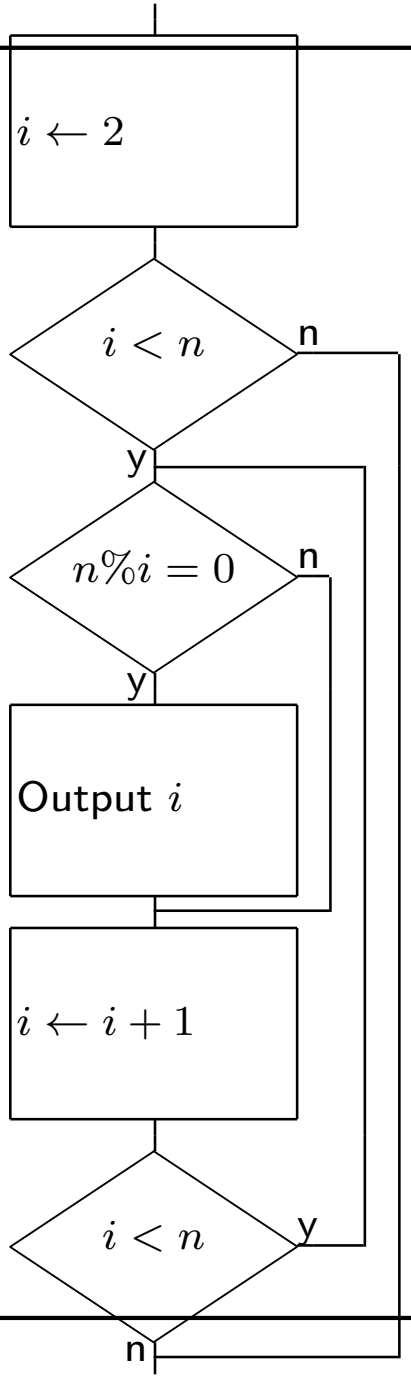
```c
/*
   This program outputs integers in the range
   2, 3, ..., n-1
*/
#include <stdio.h>

int main(void) {
   int n=0, i=0;

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

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

   return 0;
}
```



Getting there, slowly but surely...

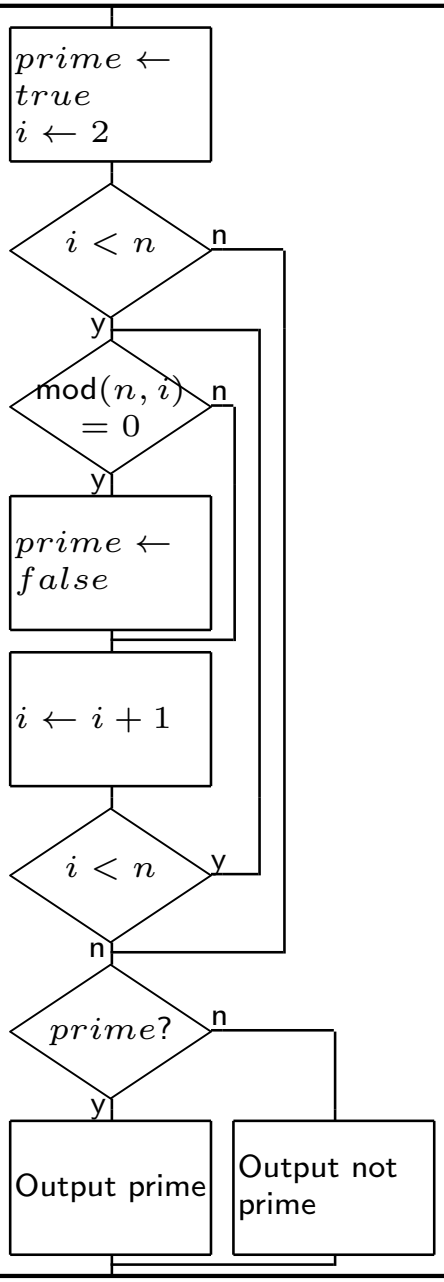
- Given $n > 1$, find all divisors of n, apart from 1 and itself.

```
/*
   This program outputs divisors of n,
   apart from 1 and itself.
*/
#include <stdio.h>

int main(void) {
   int n=0, i=0;

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

   for (i = 2; i < n; i++) {
      if (n%i == 0) { /* n divisible by i? */
         printf("%d\n", i);
      }
   }

   return 0;
}
```



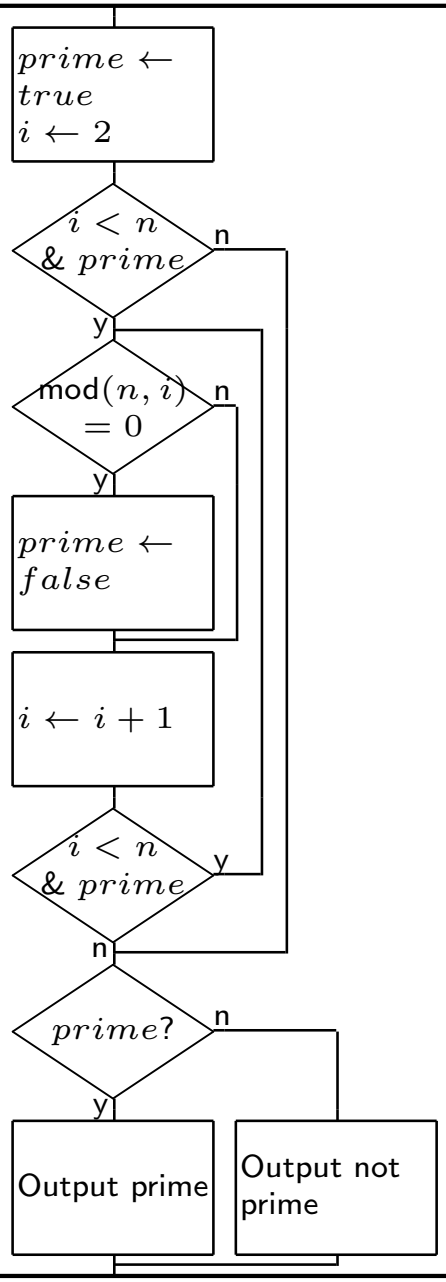
A solution that works...

```c
/*
   This program determines if n is prime.
*/
#include <stdio.h>
#include <stdbool.h>

int main(void) {
   int n=0, i=0;
   bool prime=true; /* prime by default */

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

   for (i = 2; i < n; i++) {
      if (n%i == 0) {
         prime=false;
      }
   }

   if (prime) { /* if (prime == true) */
      printf("%d is prime\n", n);
   } else {
      printf("%d is not prime\n", n);
   }

   return 0;
}
```



Better solution... early stopping

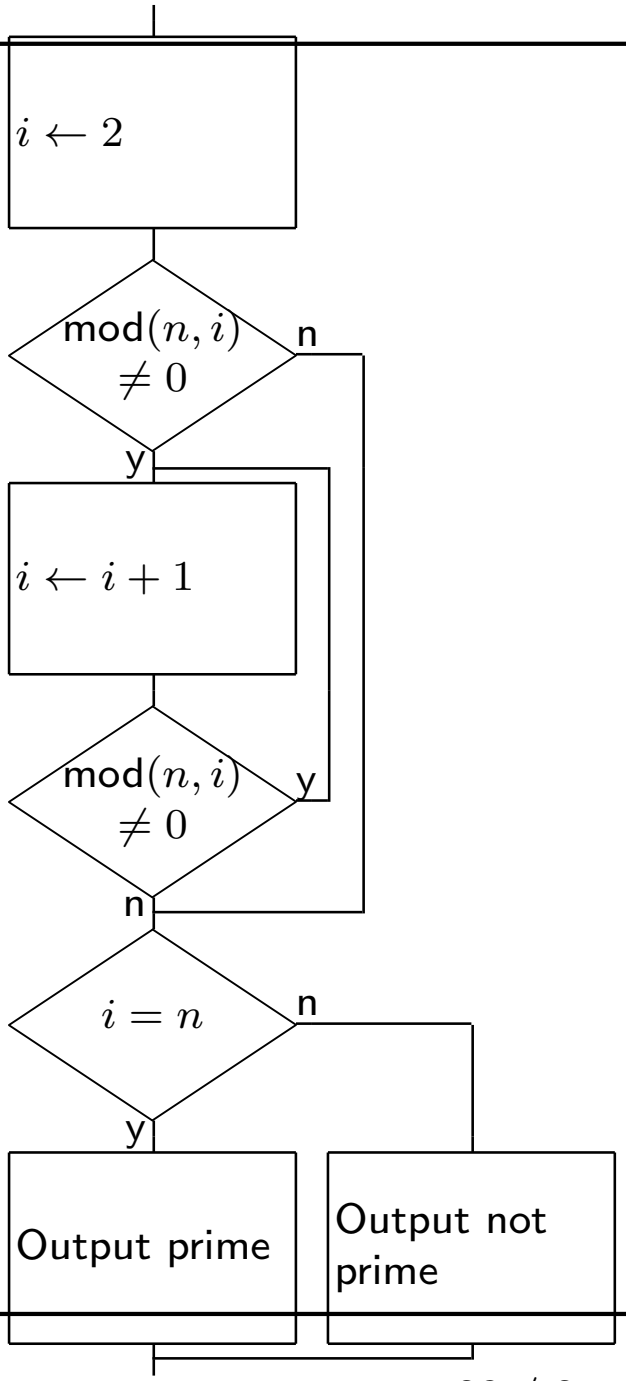
```c
#include <stdio.h>
#include <stdbool.h>

int main(void) {
   int n=0, i=0;
   bool prime=true;

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

   i = 2;
   while ((i < n) && prime)
      if (n%i == 0) {
         prime=false;
      }
      i++;
   }

   if (prime) {
      printf("%d is prime\n", n);
   } else {
      printf("%d is not prime\n", n);
   }

   return 0;
}
```



An even better solution...

- Given $n > 1$, determine if n is prime.

```c
/*
   This program determines if n is prime.
*/
#include <stdio.h>

int main(void) {
   int n=0, i=0;

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

   i = 2;
   while (n%i != 0) {
      i++;
   }

   if (i == n) {
      printf("%d is prime\n", n);
   } else {
      printf("%d is not prime\n", n);
   }

   return 0;
}
```



Lecture Summary

- Consider the different ways of handling multiple user input
- Assess the suitability of different loop constructs
- Consider loop conditions: what makes it continue?
- Consider a generalized statement body
- Apply timeline tracing for loops
- Apply incremental problem solving
  - It is an art you need to master
  - Invest enough time to hone your problem solving skills
  - Incremental development results in less time spent on debugging the program

Make it work, make it right, make it fast