

Chapter 13

Control Structures

Control Structures

Conditional

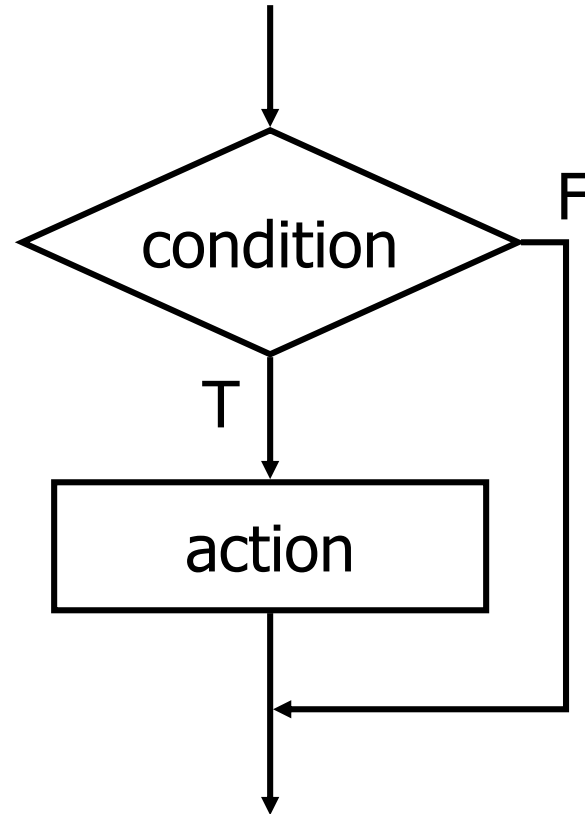
- making a decision about which code to execute, based on evaluated expression
- `if`
- `if-else`
- `switch`

Iteration

- executing code multiple times, ending based on evaluated expression
- `while`
- `for`
- `do-while`

If

```
if (condition)  
    action;
```



Condition is a C expression,
which evaluates to TRUE (non-zero) or FALSE (zero).
Action is a C statement,
which may be simple or compound (a block).

Example If Statements

```
if (x <= 10)
    y = x * x + 5;
```

```
if (x <= 10) {
    y = x * x + 5;
    z = (2 * y) / 3;
}
```

compound statement;
both executed if $x \leq 10$

```
if (x <= 10)
    y = x * x + 5;
z = (2 * y) / 3;
```

only first statement is conditional;
second statement is
always executed

More If Examples

```
if (0 <= age && age <= 11)
    kids += 1;
```

```
if (month == 4 || month == 6 ||
    month == 9 || month == 11)
    printf("The month has 30 days.\n");
```

```
if (x = 2)
    y = 5;
```

always true,
so action is ***always*** executed!

This is a common programming error (= instead of ==), not caught by compiler because it's syntactically correct.

If's Can Be Nested

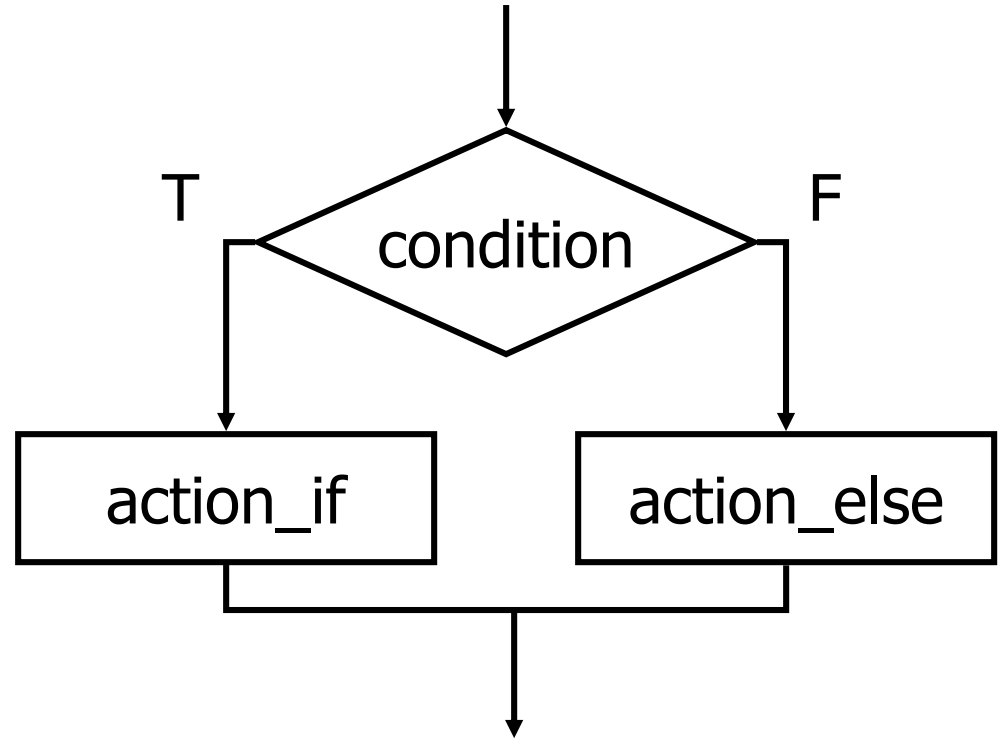
```
if (x == 3)
    if (y != 6) {
        z = z + 1;
        w = w + 2;
    }
```

is the same as...

```
if ((x == 3) && (y != 6)) {
    z = z + 1;
    w = w + 2;
}
```

If-else

```
if (condition)
    action_if;
else
    action_else;
```



Else allows choice between two mutually exclusive actions without re-testing condition.

Matching Else with If

Else is always associated with closest unassociated if.

```
if (x != 10)
  if (y > 3)
    z = z / 2;
else
  z = z * 2;
```

is the same as...

```
if (x != 10) {
  if (y > 3)
    z = z / 2;
else
  z = z * 2;
}
```

is NOT the same as...

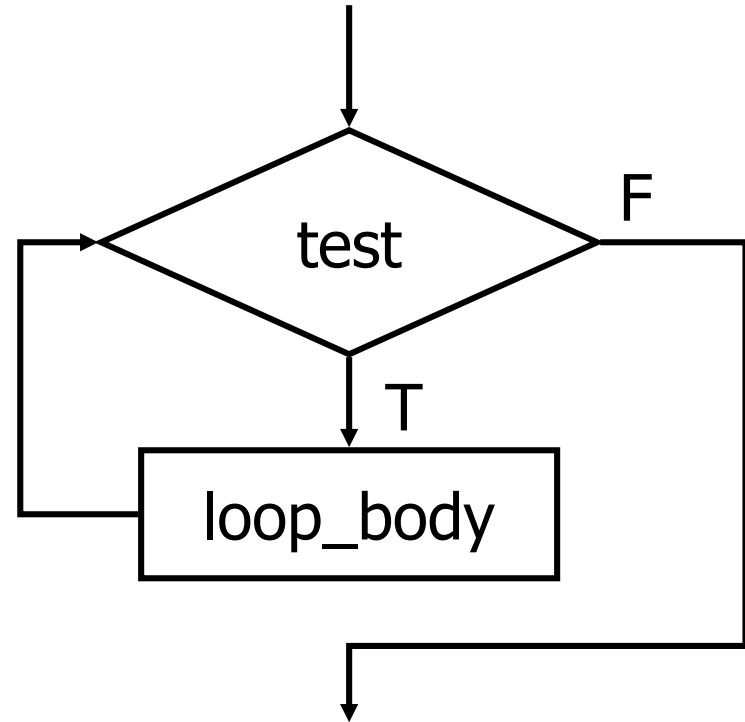
```
if (x != 10) {
  if (y > 3)
    z = z / 2;
}
else
  z = z * 2;
```


Chaining If's and Else's

```
if (month == 4 || month == 6 || month == 9 ||
    month == 11)
    printf("Month has 30 days.\n");
else if (month == 1 || month == 3 ||
        month == 5 || month == 7 ||
        month == 8 || month == 10 ||
        month == 12)
    printf("Month has 31 days.\n");
else if (month == 2)
    printf("Month has 28 or 29 days.\n");
else
    printf("Don't know that month.\n");
```

While

```
while (test)  
    loop_body;
```



Executes loop body as long as test evaluates to TRUE (non-zero).

*Note: Test is evaluated **before** executing loop body.*

Infinite Loops

The following loop will never terminate:

```
x = 0;  
while (x < 10)  
    printf("%d ", x);
```

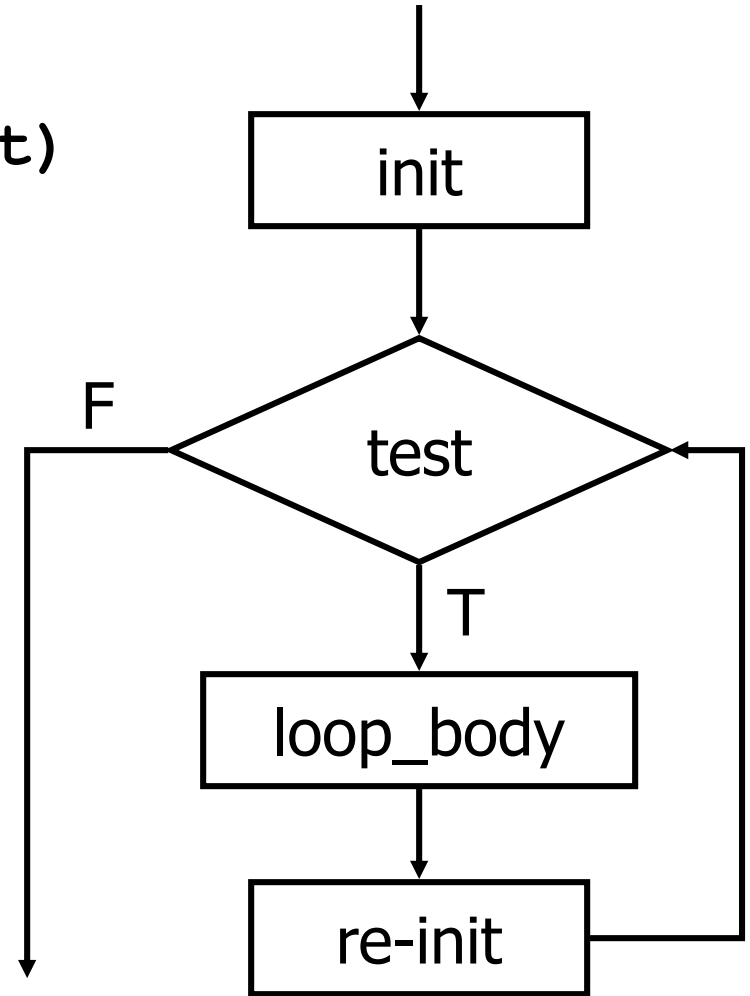
Loop body does not change condition,
so test never fails.

This is a common programming error
that can be difficult to find.

For

```
for (init; end-test; re-init)  
    statement
```

Executes loop body as long as test evaluates to TRUE (non-zero). Initialization and re-initialization code included in loop statement.



*Note: Test is evaluated **before** executing loop body.*

Example For Loops

```
/* -- what is the output of this loop? -- */
```

```
for (i = 0; i <= 10; i ++)  
    printf("%d ", i);
```

```
/* -- what does this one output? -- */
```

```
letter = 'a';  
for (c = 0; c < 26; c++)  
    printf("%c ", letter+c);
```

```
/* -- what does this loop do? -- */
```

```
numberOfOnes = 0;  
for (bitNum = 0; bitNum < 16; bitNum++) {  
    if (inputValue & (1 << bitNum))  
        numberOfOnes++;  
}
```

Nested Loops

Loop body can (of course) be another loop.

```
/* print a multiplication table */
for (mp1 = 0; mp1 < 10; mp1++) {
    for (mp2 = 0; mp2 < 10; mp2++) {
        printf("%d\t", mp1*mp2);
    }
    printf("\n");
}
```

Braces aren't necessary,
but they make the code easier to read.

Another Nested Loop

The test for the inner loop depends on the counter variable of the outer loop.

```
for (outer = 1; outer <= input; outer++) {  
    for (inner = 0; inner < outer; inner++) {  
        sum += inner;  
    }  
}
```

For vs. While

In general:

For loop is preferred for **counter**-based loops.

- Explicit counter variable
- Easy to see how counter is modified each loop

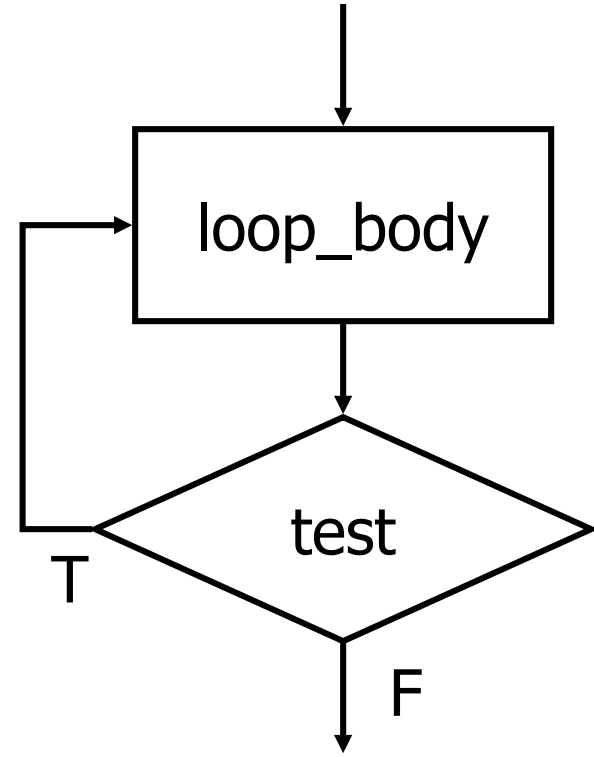
While loop is preferred for **sentinel**-based loops.

- Test checks for sentinel value.

Either kind of loop can be expressed as the other, so it's really a matter of style and readability.

Do-While

```
do  
    loop_body;  
while (test);
```



Executes loop body as long as test evaluates to TRUE (non-zero).

*Note: Test is evaluated **after** executing loop body.*

Problem Solving in C

Stepwise Refinement

- as covered in Chapter 6

...but can stop refining at a higher level of abstraction.

Same basic constructs

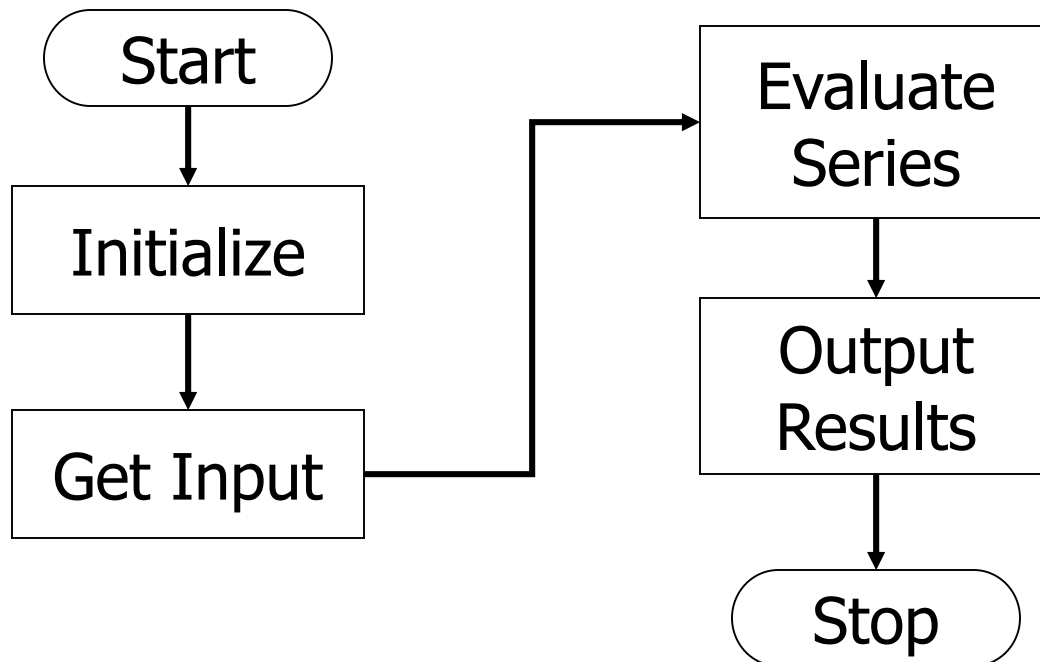
- **Sequential** -- C statements
- **Conditional** -- if-else, switch
- **Iterative** -- while, for, do-while

Problem 1: Calculating Pi

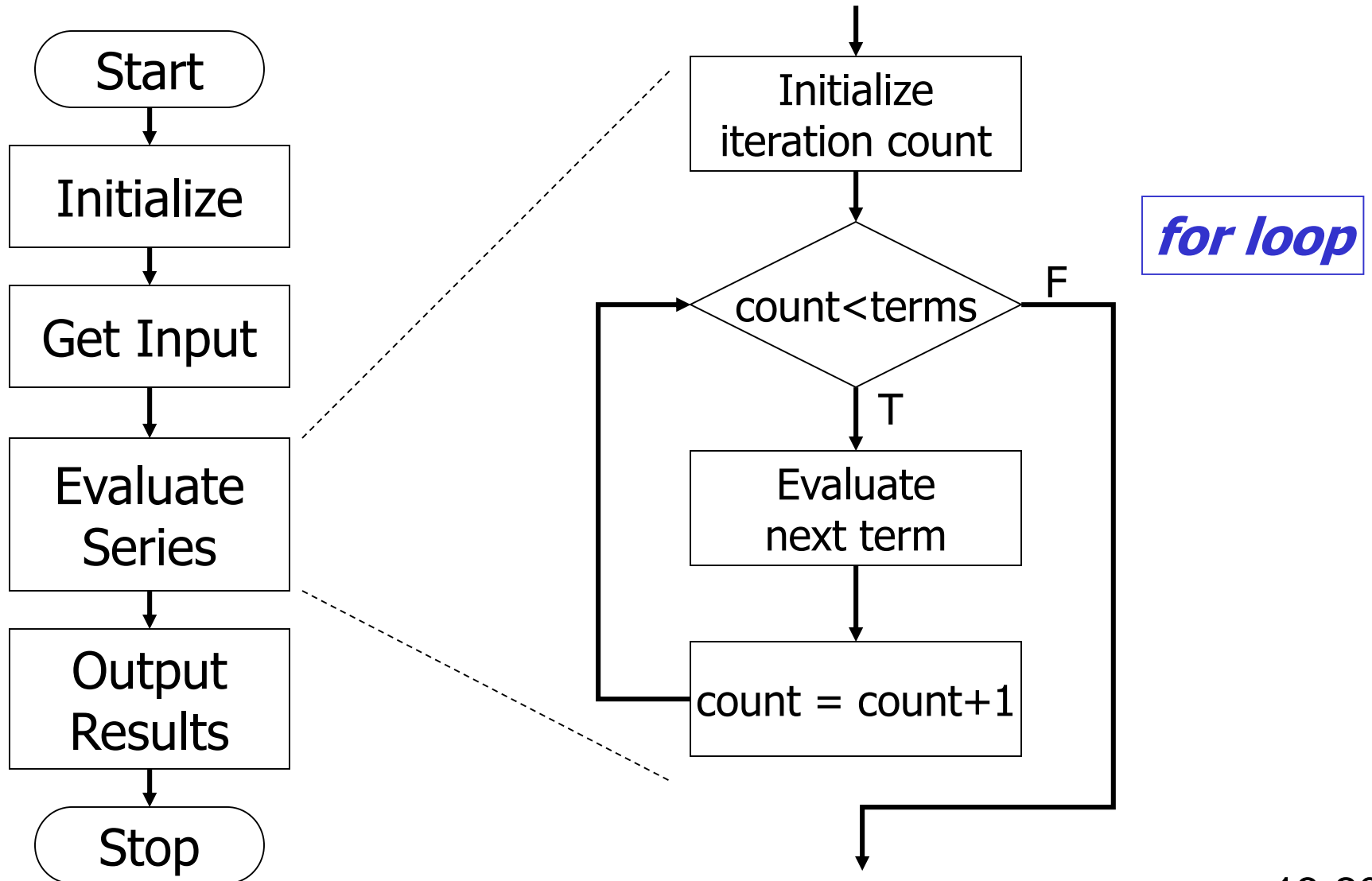
Calculate π using its series expansion.

User inputs number of terms.

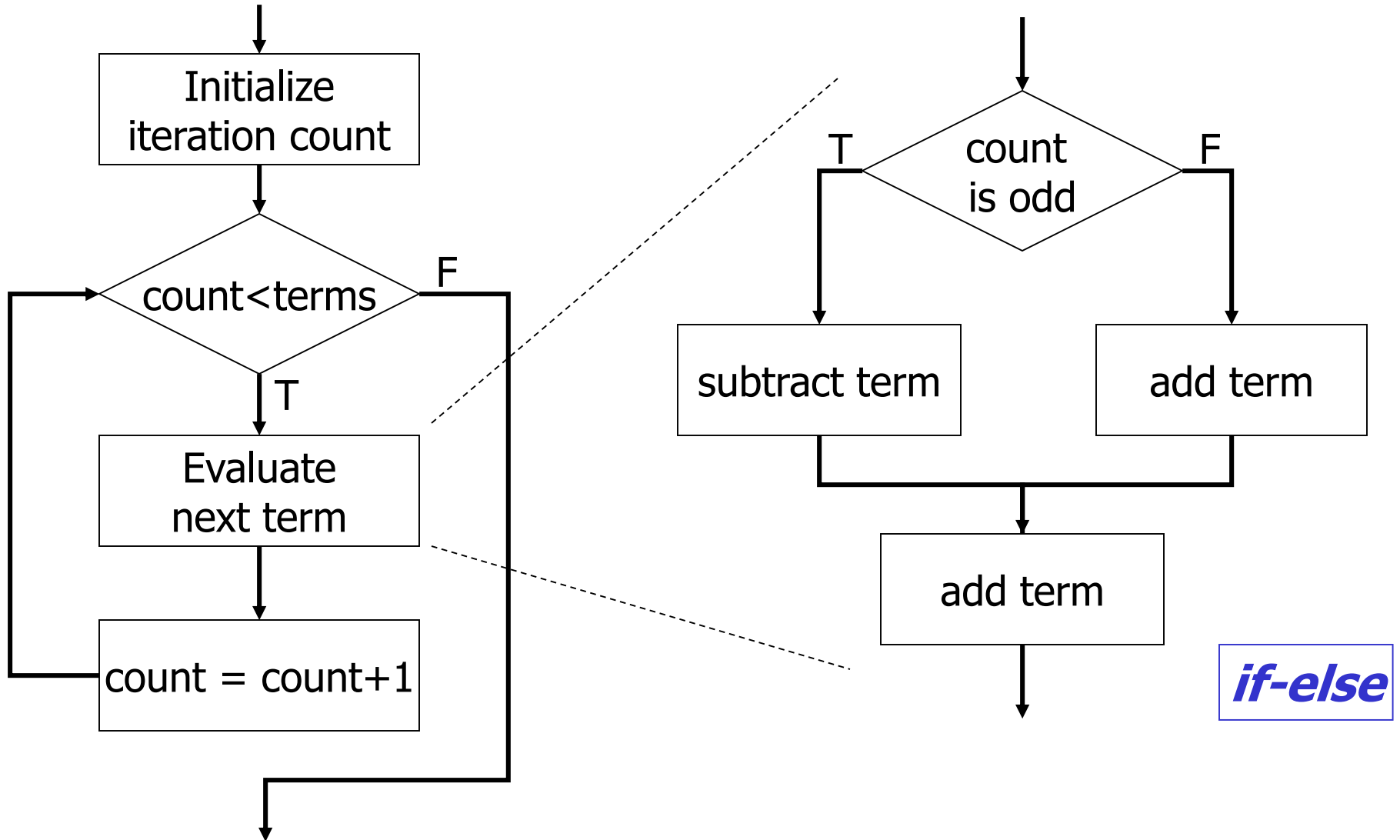
$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \dots + (-1)^{n-1} \frac{4}{2n+1} + \dots$$



Pi: 1st refinement



Pi: 2nd refinement



Pi: Code for Evaluate Terms

```
for (count=0; count < numOfTerms; count++) {  
    if (count % 2) {  
        /* odd term -- subtract */  
        pi -= 4.0 / (2 * count + 1);  
    }  
    else {  
        /* even term -- add */  
        pi += 4.0 / (2 * count + 1);  
    }  
}
```

Note: Code in text is slightly different,
but this code corresponds to equation.

Pi: Complete Code

```
#include <stdio.h>

main() {
    double pi = 0.0;
    int numOfTerms, count;

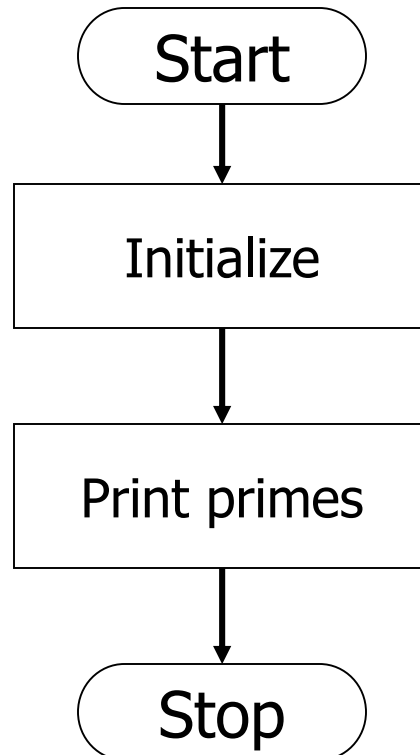
    printf("Number of terms (must be 1 or larger) : ");
    scanf("%d", &numOfTerms);

    for (count=0; count < numOfTerms; count++) {
        if (count % 2) {
            pi -= 4.0 / (2 * count + 1); /* odd term -- subtract */
        }
        else {
            pi += 4.0 / (2 * count + 1); /* even term -- add */
        }
    }
    printf("The approximate value of pi is %f\n", pi);
    return 0;
}
```

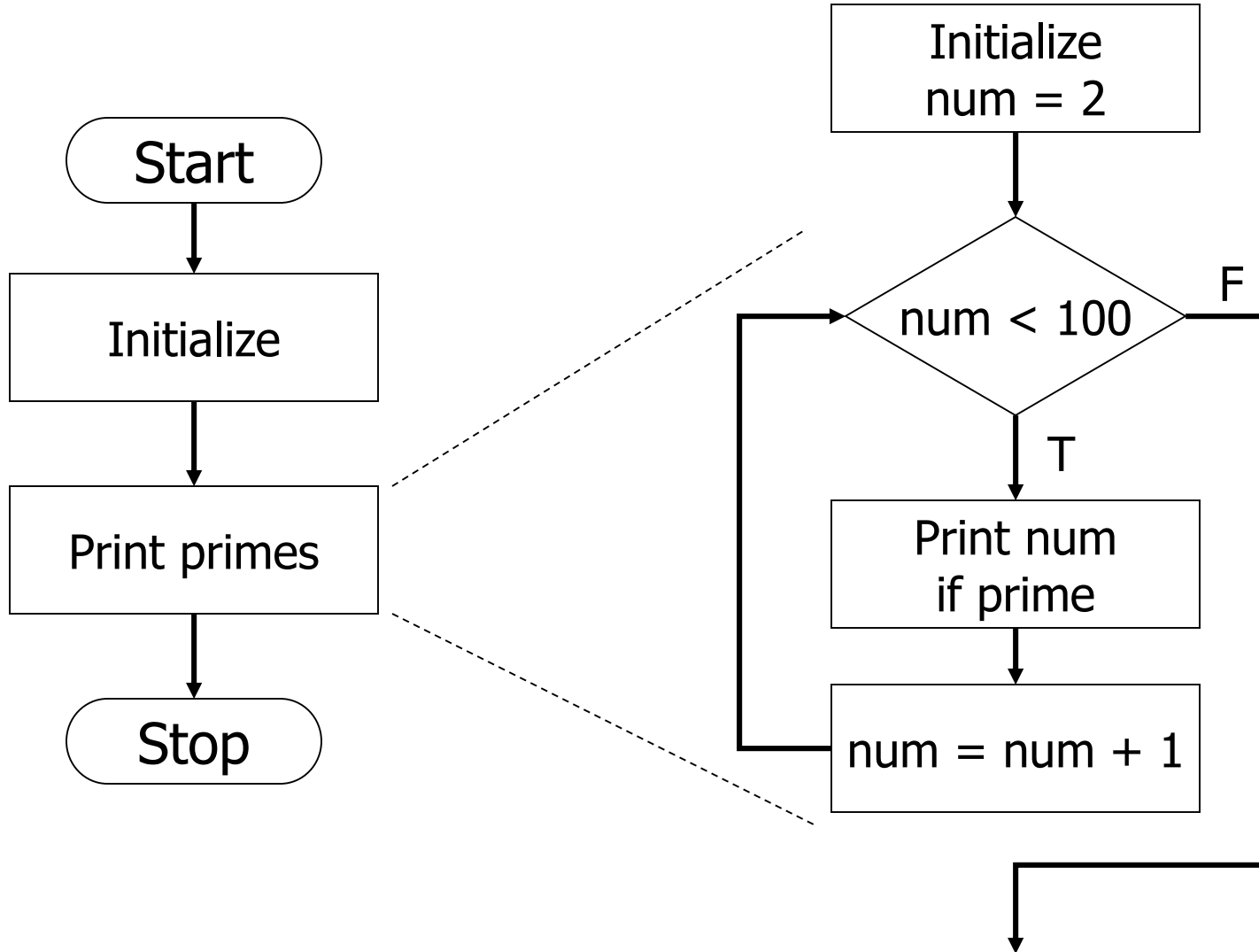
Problem 2: Finding Prime Number

Print all prime numbers less than 100.

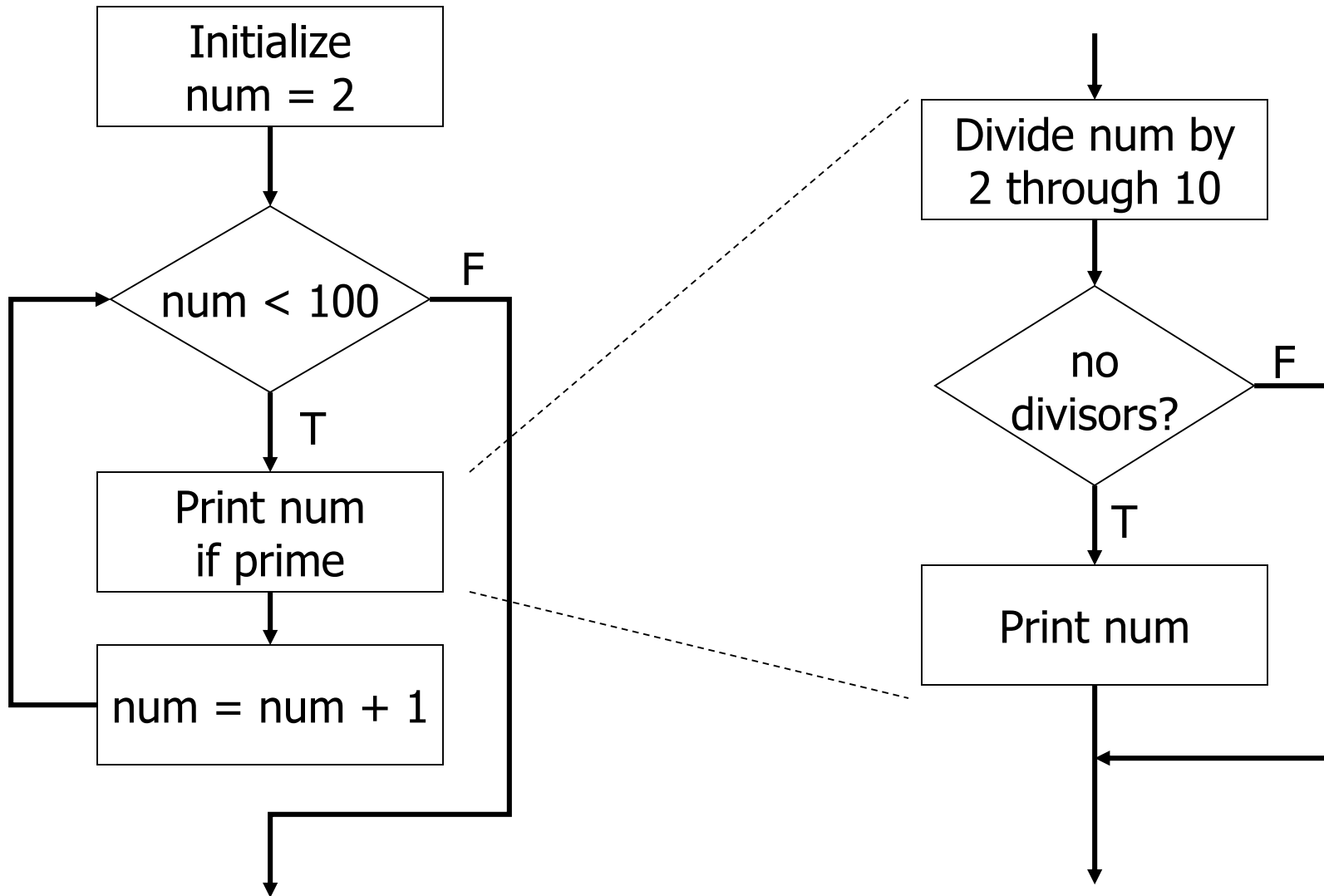
- A number is prime if its only divisors are 1 and itself.
- All non-prime numbers less than 100 will have a divisor between 2 and 10.



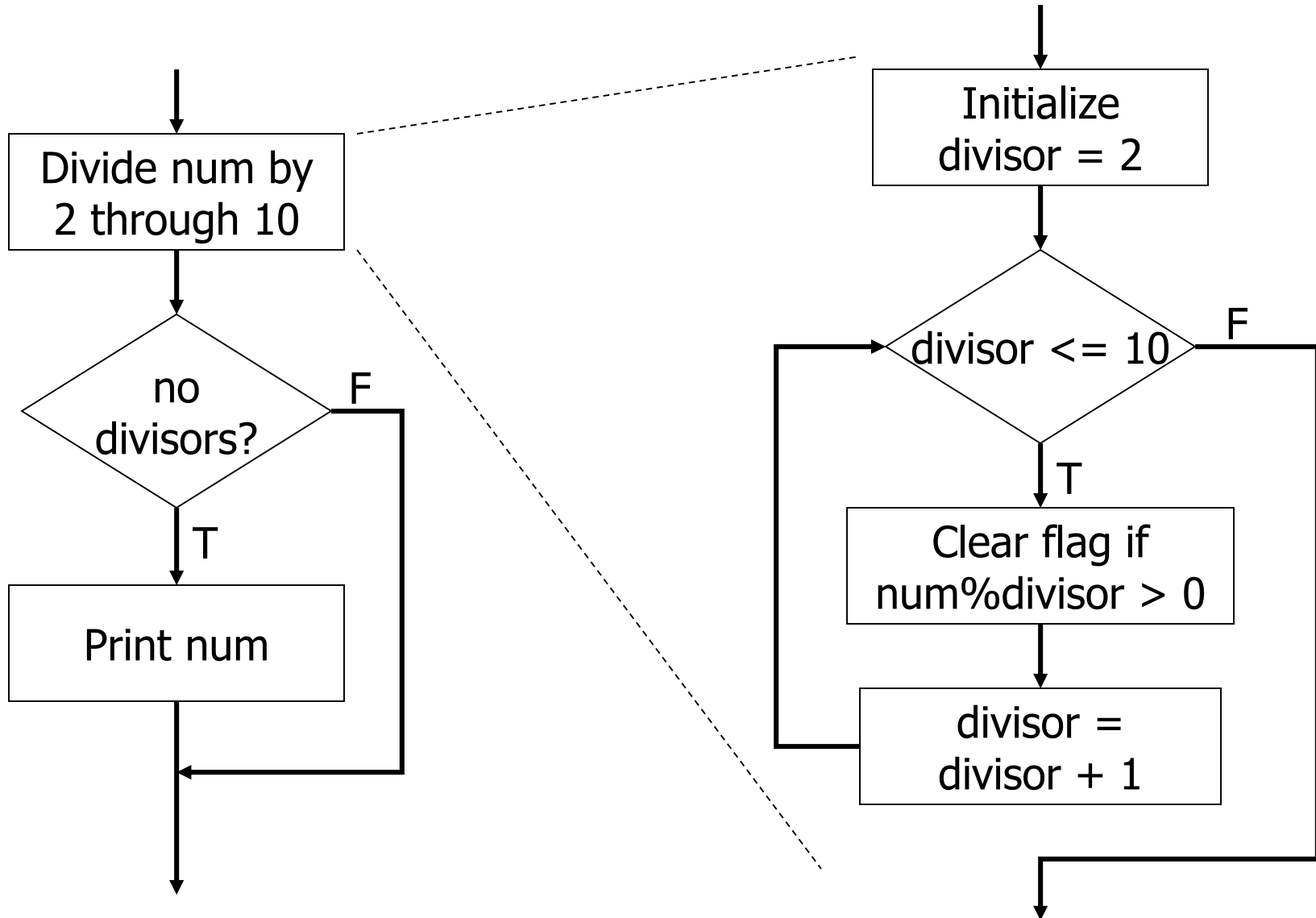
Primes: 1st refinement



Primes: 2nd refinement



Primes: 3rd refinement



Primes: Using a Flag Variable

To keep track of whether number was divisible, we use a "flag" variable.

- Set `prime = TRUE`, assuming that this number is prime.
- If any divisor divides number evenly, set `prime = FALSE`.
 - Once it is set to `FALSE`, it stays `FALSE`.
- After all divisors are checked, number is prime if the flag variable is still `TRUE`.

Use macros to help readability.

```
#define TRUE 1  
#define FALSE 0
```

Primes: Complete Code

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0

main () {
    int num, divisor, prime;

    /* start with 2 and go up to 100 */
    for (num = 2; num < 100; num ++ ) {

        prime = TRUE; /* assume num is prime */
        /* test whether divisible by 2 through 10 */
        for (divisor = 2; divisor <= 10; divisor++)
            if (((num % divisor) == 0) && (num != divisor))
                prime = FALSE; /* not prime */

        if (prime) /* if prime, print it */
            printf("The number %d is prime\n", num);
    }
}
```

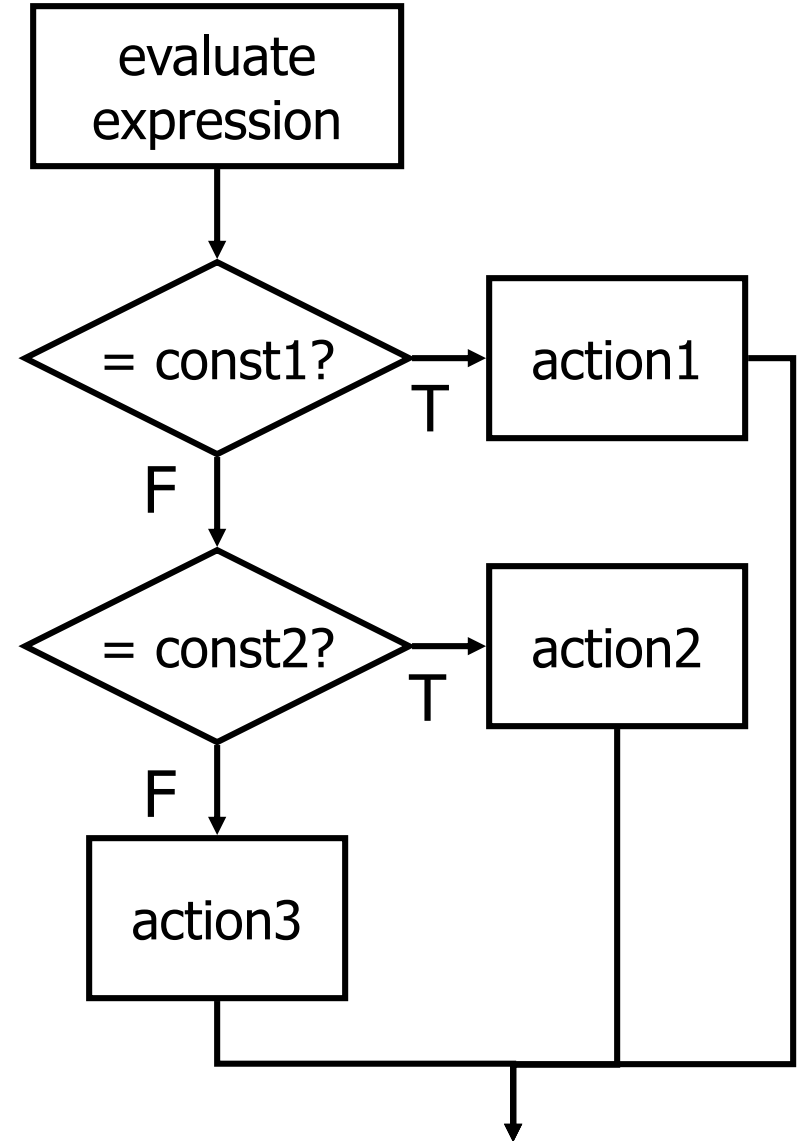
Optimization: Could put
a break here to avoid some work.
(Section 13.5.2)

Continue here

Switch

```
switch (expression) {  
  case const1:  
    action1; break;  
  case const2:  
    action2; break;  
  default:  
    action3;  
}
```

*Alternative to long if-else chain.
If break is not used, then
case "falls through" to the next.*



Switch Example

```
/* same as month example for if-else */
switch (month) {
    case 4:
    case 6:
    case 9:
    case 11:
        printf("Month has 30 days.\n");
        break;
    case 1:
    case 3:
        /* some cases omitted for brevity...*/
        printf("Month has 31 days.\n");
        break;
    case 2:
        printf("Month has 28 or 29 days.\n");
        break;
    default:
        printf("Don't know that month.\n");
}
```

More About Switch

Case expressions must be constant.

```
case i: /* illegal if i is a variable */
```

If no break, then next case is also executed.

```
switch (a) {  
    case 1:  
        printf("A");  
    case 2:  
        printf("B");  
    default:  
        printf("C");  
}
```

**If a is 1, prints "ABC".
If a is 2, prints "BC".
Otherwise, prints "C".**

Problem 3: Searching for Substring

Have user type in a line of text (ending with linefeed) and print the number of occurrences of "the".

Reading characters one at a time

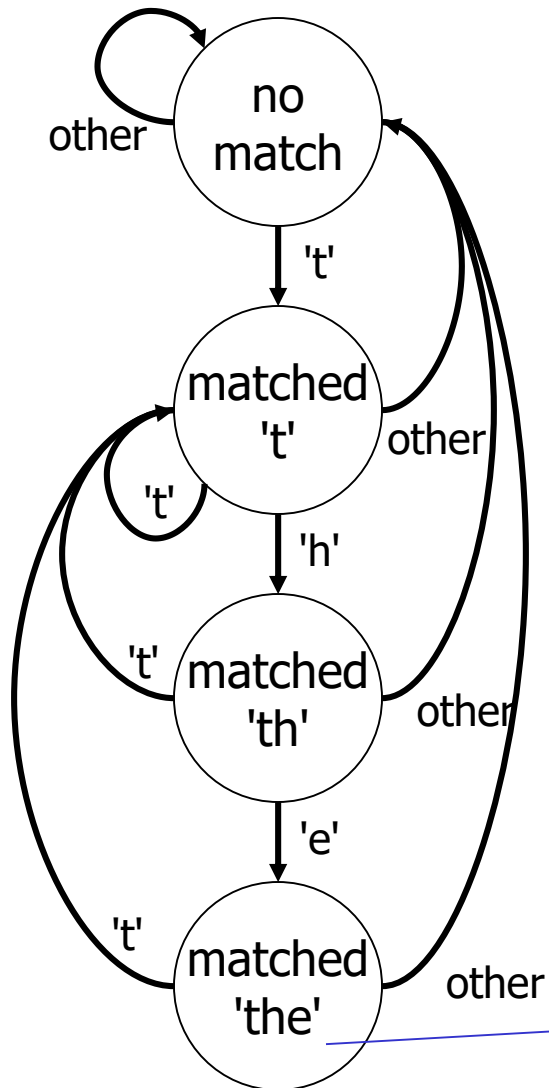
- Use the `getchar()` function -- returns a single character.

Don't need to store input string;

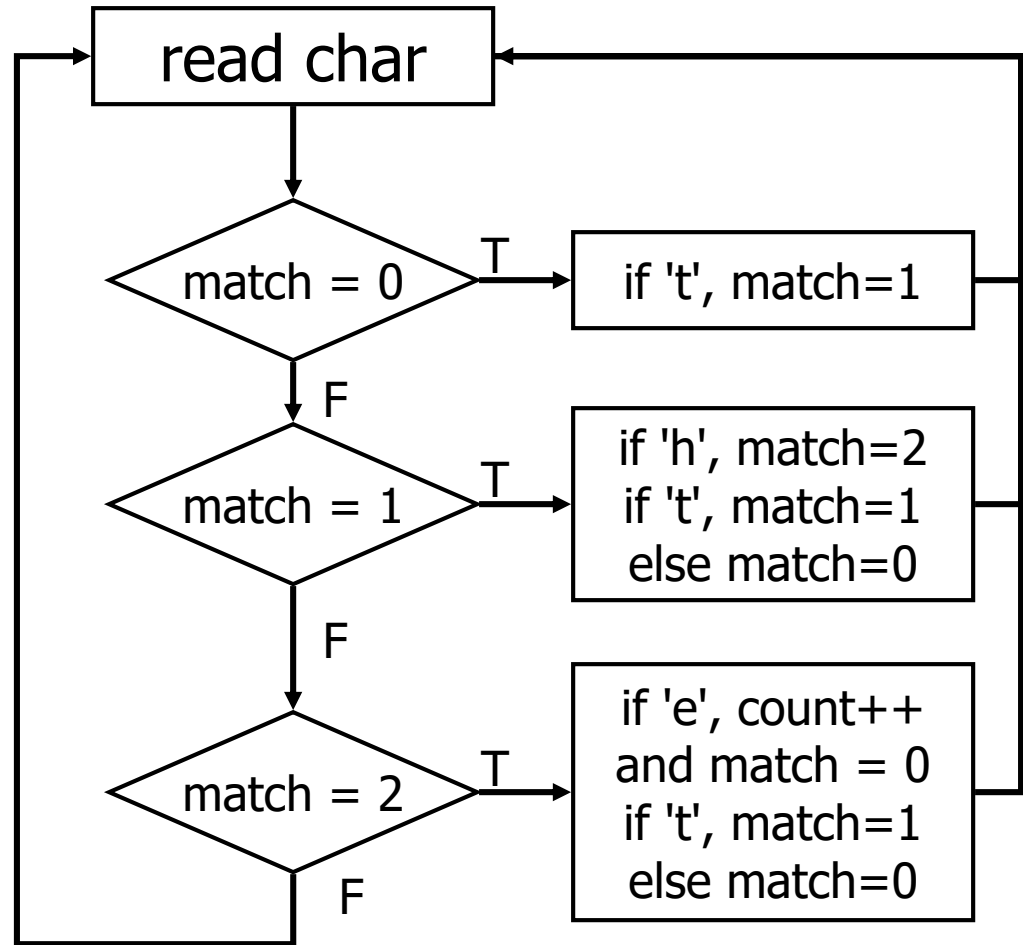
look for substring as characters are being typed.

- Similar to state machine:
based on characters seen, move toward success state or move back to start state.
- **Switch statement** is a good match to state machine.

Substring: State machine to flow chart



increment
count



Substring: Code (Part 1)

```
#include <stdio.h>

main() {
    char key;          /* input character from user */
    int match = 0;    /* keep track of characters matched */
    int count = 0;    /* number of substring matches */

    /* Read character until newline is typed */
    while ((key = getchar()) != '\n') {

        /* Action depends on number of matches so far */
        switch (match) {

            case 0: /* starting - no matches yet */
                if (key == 't')
                    match = 1;
                break;
```

Substring: Code (Part 2)

```
case 1: /* 't' has been matched */
    if (key == 'h')
        match = 2;
    else if (key == 't')
        match = 1;
    else
        match = 0;
    break;
```

Substring: Code (Part 3)

```
case 2: /* 'th' has been matched */
    if (key == 'e') {
        count++; /* increment count */
        match = 0; /* go to starting point */
    }
    else if (key == 't') {
        match = 1;
    }
    else
        match = 0;
    break;
}
}
printf("Number of matches = %d\n", count);
}
```

Break and Continue

break ;

- used *only* in switch statement or iteration statement
- passes control out of the “smallest” (loop or switch) statement containing it to the statement immediately following
- usually used to exit a loop before terminating condition occurs (or to exit switch statement when case is done)

continue ;

- used only in iteration statement
- terminates the execution of the loop body for this iteration
- loop expression is evaluated to see whether another iteration should be performed
- if `for` loop, also executes the re-initializer

Example

What does the following loop do?

```
for (i = 0; i <= 20; i++) {  
    if (i%2 == 0) continue;  
    printf("%d ", i);  
}
```

What would be an easier way to write this?

What happens if **break** instead of **continue**?

Looking Ahead

A glimpse of what is coming.

- **Functions**
- **Pointers, arrays and strings**
- **C file I/o**

Slides after this skipped: LC-3 Implementation

Slides after this skipped

We will come back to these after we have seen LC-3

Generating Code for If-Else

```
if (x) {
    y++;
    z--;
}
else {
    y--;
    z++;
}

LDR    R0, R5, #0
BRz    ELSE
; x is not zero
LDR    R1, R5, #-1 ; incr y
ADD    R1, R1, #1
STR    R1, R5, #-1
LDR    R1, R5, #02 ; decr z
ADD    R1, R1, #1
STR    R1, R5, #-2
JMP    DONE ; skip else code
; x is zero
ELSE   LDR    R1, R5, #-1 ; decr y
      ADD    R1, R1, #-1
      STR    R1, R5, #-1
      LDR    R1, R5, #-2 ; incr z
      ADD    R1, R1, #1
      STR    R1, R5, #-2
DONE   ... ; next statement
```

Generating Code for While

```
x = 0;
while (x < 10) {
    printf("%d ", x);
    x = x + 1;
}
```

LOOP

```
AND    R0, R0, #0
STR    R0, R5, #0 ; x = 0
; test
LDR    R0, R5, #0 ; load x
ADD    R0, R0, #-10
BRzpl DONE
; loop body
LDR    R0, R5, #0 ; load x
...
<printf>
...
ADD    R0, R0, #1 ; incr x
STR    R0, R5, #0
JMP    LOOP ; test again

DONE ; next statement
```

Generating Code for For

```
for (i = 0; i < 10; i++)  
    printf("%d ", i);
```

This is the same
as the while example!

```
                                ; init  
                                AND    R0, R0, #0  
                                STR    R0, R5, #0 ; i = 0  
                                ; test  
LOOP    LDR    R0, R5, #0 ; load i  
                                ADD    R0, R0, #-10  
                                BRz    DONE  
                                ; loop body  
                                LDR    R0, R5, #0 ; load i  
                                ...  
                                <printf>  
                                ...  
                                ; re-init  
                                ADD    R0, R0, #1 ; incr i  
                                STR    R0, R5, #0  
                                JMP    LOOP ; test again  
  
DONE    ; next statement
```