# Chapter 3
# **Digital Logic Structures**

Original slides from Gregory Byrd, North Carolina State University

Modified slides by C. Wilcox, S. Rajopadhye
Colorado State University

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

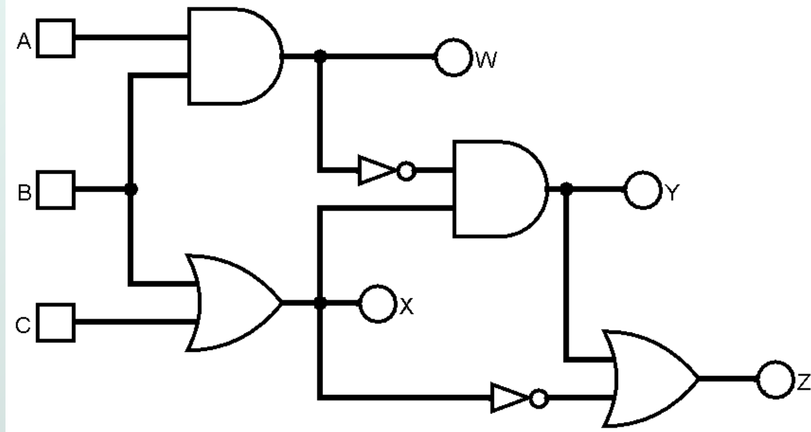
## Computing Layers

Problems
Algorithms
Language
Instruction Set Architecture
Microarchitecture
Circuits ←
Devices

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Combinational Logic

- Cascading set of logic gates




What is the truth table?



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Truth Table (from circuit)

- Truth table for circuit on previous slide

| A | B | C | W | X | Y | Z |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## Logisim Simulator

- Logic simulator: allows interactive design and layout of circuits with AND, OR, and NOT gates
- Simulator web page (linked on class web page)
  http://ozark.hendrix.edu/~burch/logisi
- Overview, tutorial, downloads, etc.
- Windows or Linux operating systems
- Logisim demonstration

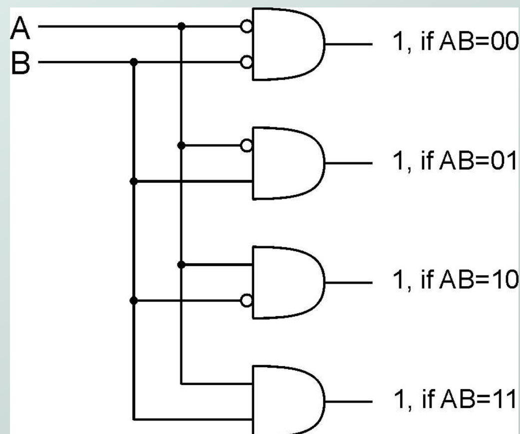


Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## Decoder

- *n* inputs, $2^n$ outputs
  - exactly one output is 1 for each possible input pattern

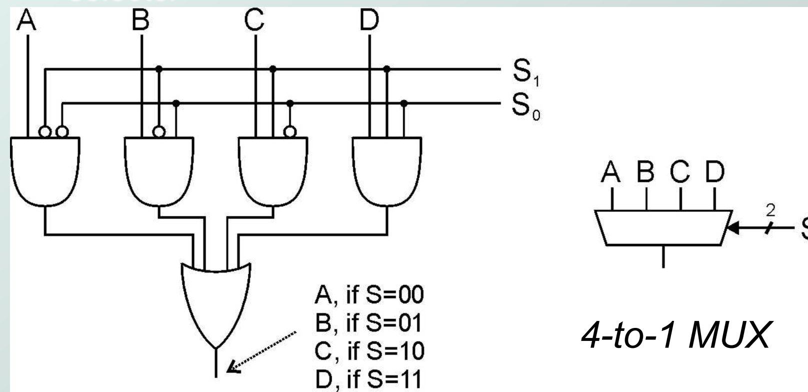
*2-bit decoder*



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Multiplexer (MUX)

- *n*-bit selector and $2^n$ inputs, one output
  - output equals one of the inputs, depending on selector

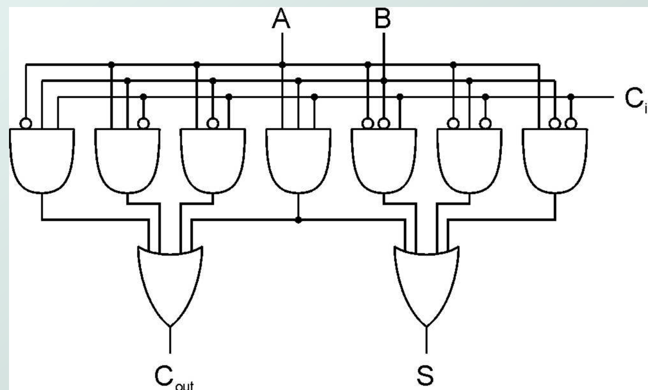





Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Full Adder

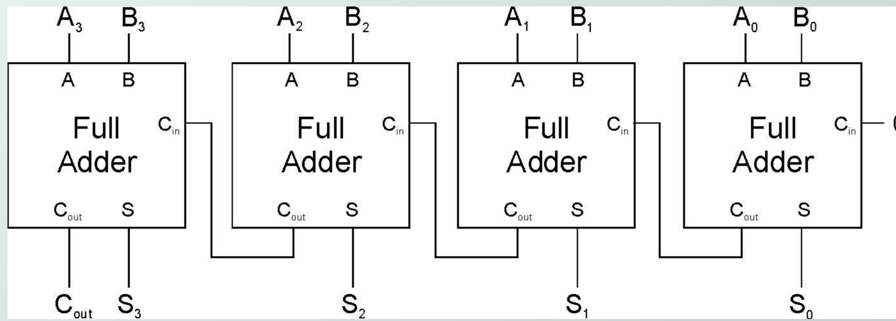
- Add two bits and carry-in, produce one-bit sum and carry-out.




| A | B | $C_{in}$ | S | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Four-bit Adder



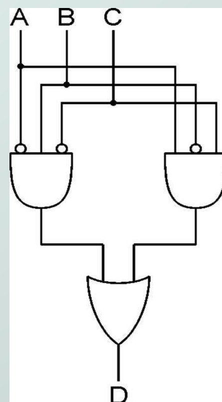



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Logical Completeness

- Can implement ANY truth table with combo of AND, OR, NOT gates.

| A | B | C | D |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |




1. AND combinations that yield a "1" in the truth table.
2. OR the results of the AND gates.

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Truth Table (to circuit)

- How do we design a circuit for this?

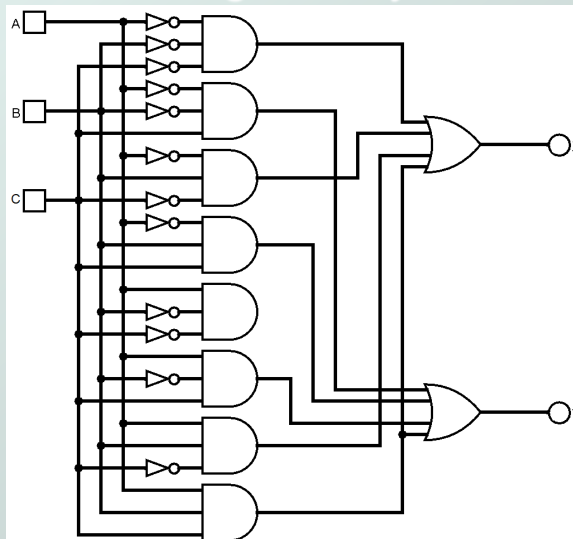
| A | B | C | X | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Programmable Logic Array

- Front end is a input decode
- Back end selects outputs
- Not necessarily minimal circuit!
- Logic arrays are prebuilt



# Chapter 13
# Control Structures

Original slides from Gregory Byrd, North Carolina State University

Modified slides by C Wilcox, S Rajopadhye
Colorado State University

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## Control Structures

- Conditional
  - making a decision about which code to execute, based on evaluated expression

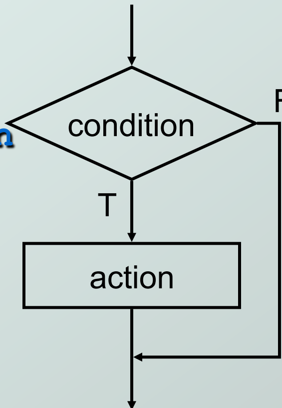
  **`if`**
  **`if-else`**
  **`switch`**
- Iteration
  - executing code multiple times, ending based on evaluated expression

  **`while`**
  **`for`**
  **`do-while`**

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# If

```
if (condition)
    statement;//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).*



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# 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 <= 10
- ```
  if (x <= 10)
     y = x * x + 5;
  z = (2 * y) / 3;
  ```
  only first statement is conditional; second statement is ***always*** executed
  ```

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## 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!

A common programming error (= instead ==), not caught by compiler because it is syntactically correct.



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## If s Can Be Nested

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

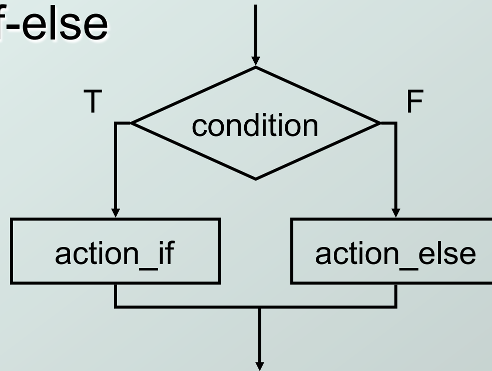
**is the same as...**

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

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## If-else

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




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



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## 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;
```

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Chaining If s and Else s

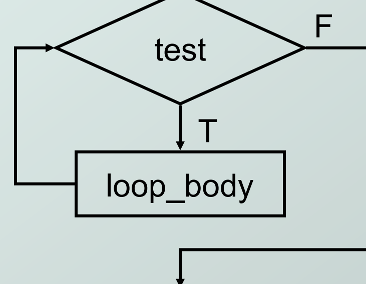
```
if      (month == 4 || month == 6 ||
         month == 9 || month == 11)
    printf("Month has 30 days.\n");
else if (month == 1 || month == 3  ||
         month == 5 || month == 7  ||
         month == 8 || month == 10 ||
         month == 12)
    printf("Month has 31 days.\n");
else if (month ==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



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Iteration 1: 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.*

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Infinite Loops

- The following loop will never terminate:

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

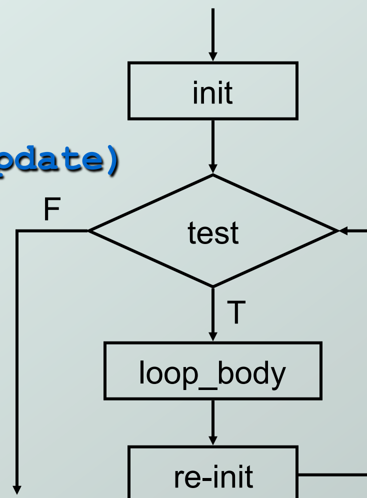
- Loop body does not change condition, so test never fails.
- This is a common programming error that can be difficult to find.



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# For

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




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

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

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# 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++;
  }
```



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# 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 make the code more readable. Also avoids bugs when editing

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## 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;
    }
  }
```



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## For vs. While

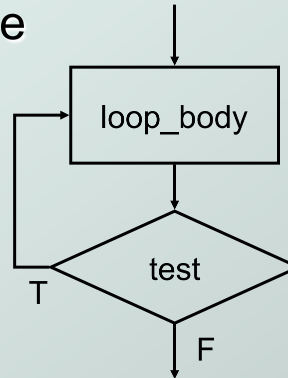
In general:

- 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.

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# 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. So* loop_body *is executed* ***at least once***



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Problem Solving in C

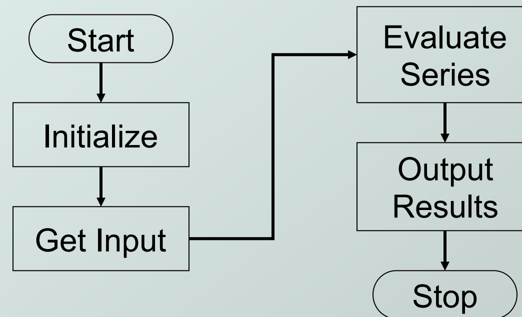
- Stepwise Refinement
  - as covered in Chapter 6
- ...but can stop refining at a higher level of abstraction.
- Same basic constructs
  - Sequential -- C statements
  - Conditional -- if-else, switch
  - Iterative -- while, for, do-while

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Problem 1: Calculating Pi

- Calculate π using its series expansion. User inputs number of terms.

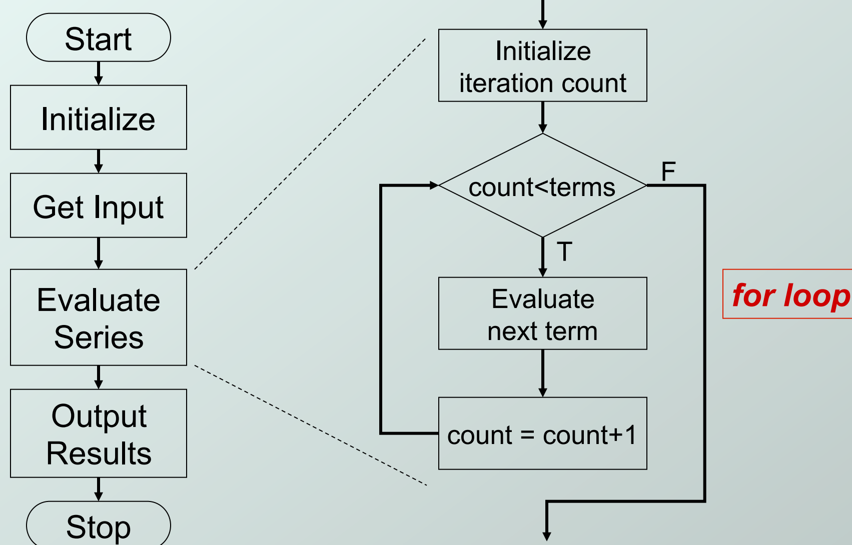
$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \cdots + (-1)^{n-1}\frac{4}{2n+1} + \cdots$$





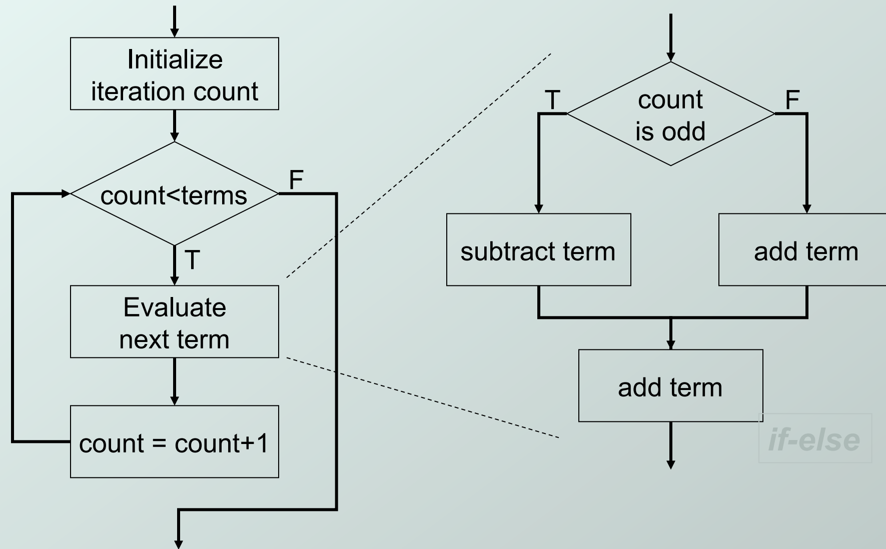

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Pi: 1st refinement



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Pi: 2nd refinement






Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Pi: Code for “Evaluate next Term”

```
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.

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Pi: Complete Code

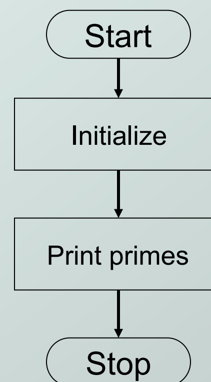
```
#include <stdio.h>
int main(int argc, char *argv[]) {
    double pi = 0.0;
    int numOfTerms, count;
    printf("Number of terms (must be 1 or larger) : ");
    scanf("%d", &numOfTerms);
    for (count=0; count < numOfTerms; count++) {
    if (count % 2) {
      pi -= 4.0 / (2 * count + 1); // odd term, subtract
    }
    else {
      pi += 4.0 / (2 * count + 1); // even term, add
     }
    }
    printf("The approximate value of pi is %f\n", pi);
  }
```

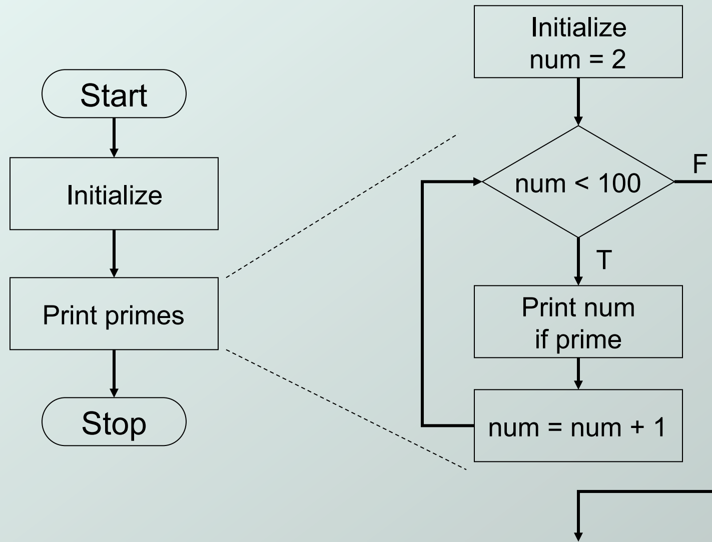


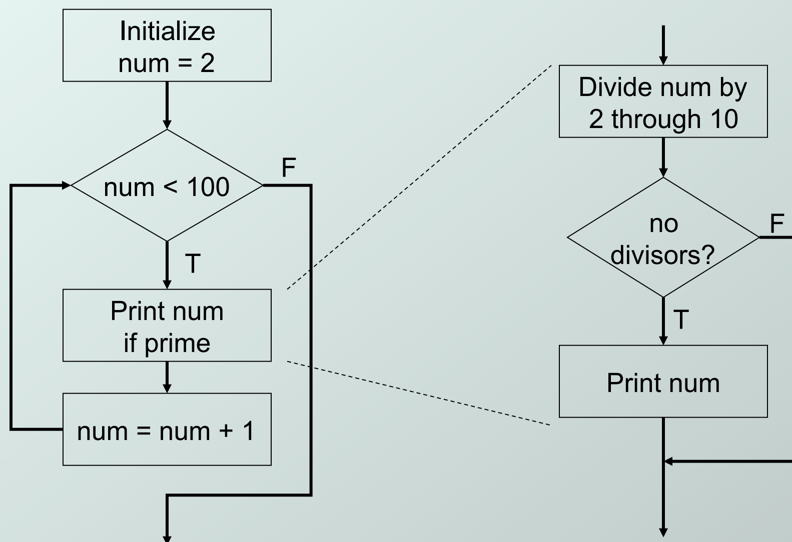
Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

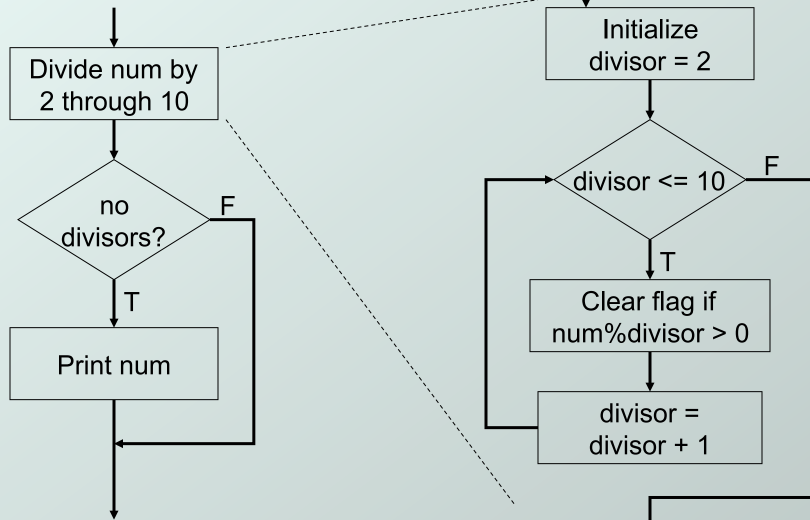
# Problem 2: Finding Prime Numbers

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




Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.
Primes: 1st refinement
Start
Initialize
Print primes
Stop
Initialize
num = 2
num < 100
F
T
Print num
if prime
num = num + 1
CS270 - Fall 2013 - Colorado State University
37


Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.
Primes: 2nd refinement
Initialize
num = 2
num < 100
F
T
Print num
if prime
num = num + 1
Divide num by
2 through 10
no
divisors?
F
T
Print num
CS270 - Fall 2013 - Colorado State University
38




Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Primes: Using a Flag Variable

- To keep track of whether number was divisible, we use a "flag" variable.
  - Set prime = TRUE, assuming that number is prime.
  - If a 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
```

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# Primes: Complete Code

```
#include <stdio.h>
#define TRUE  1
#define FALSE 0
  int main (int argc, char*argv[]) {
    int num, divisor, prime;
    /* start with 2 and go up to 100 */
    for (num = 2; num < 100; num ++ ) {
      prime = TRUE;  /* assume 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);
    }
  }
```

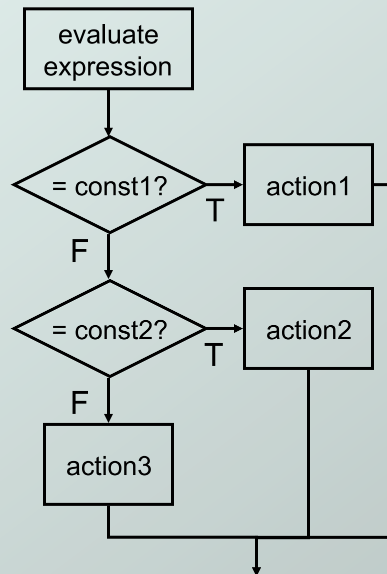
Optimization: Could put a break here to avoid some work. (Section 13.5.2)



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

# 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.*

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## 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:
    …
      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");
}
```



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## 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".