Flow of Control - Part 1
Expressions and Selection Statements

Topics

- Math and testing operators
- Various Kinds Of Flow Of Control

Constants - Revisited

- It's a good idea to name values to prevent "magic" values showing up in your code
- Use `const` declaration to state that value cannot change after assignment
- Examples:

```
const double PI = 3.14159;
const int LIMIT = 15;
```

Type Compatibility

- Need to be careful if you mix types and values on assignment statements or arithmetic expressions
- When working with

  `A operand B`

  where operand may be +, -, *, /, or %
  - if `A` or `B` is `double` or `float`, the result will be `double` or `float`

Type Compatibility

- Try the following examples to learn how C works:
  - 3 + 4.4 =
  - 2.2 * 3 =
  - 2.2 * 3.0 =
  - 2 * 3 =
  - 4.5 * 2 =
  - 9 * 2 =

Casting Operators

- Casting is used to change a literal from one type to another
- Example:

```
float pi = 3.141692;
int truncated_pi = (int)pi;
```

- With Casting, we are not changing the data type of pi, but changing the type of the literal (3.141692) it contains

Division

- When working with

  `A / B`

- If either operand is real, then the other will be converted to a real and the result will be real
- If both are `int`, then integer division occurs and the result will be an `int`
  - modulus operator `%` yields the remainder

Division

- Examples:
  - 9 / 4 =
  - 9 % 4 =
  - 11 / 4 =
  - 11 % 4 =
  - TRICK QUESTION: `var_a * (1 / 4) =`

Division

- Examples:
  - 9 / 4 = 2
  - 9 % 4 = 1
  - 11 / 4 = 2
  - 11 % 4 = 3
  - TRICK QUESTION: `var_a * (1 / 4)` = 0

Compounds or Shortcuts

- c = c + 2 can also be written as c +=2
- This applies to +, -, *, /,%, for example:
  a*=4 is the same as a=a*4
- With increment/decrement, if the right side of = is 1, you can use: a++, or c--
- a++ is the same as a=a+1
- ++ or − are referred to as compound
- You can use a++ or ++a but they have different meanings. Pay attention to the next slide

Compounds

- In C we can have:
  a=2
  a++ // a is now 3
  c=a++ //c is 3, a is 4
- In the above example, the assignment:
  c=a
  happens first then a is incremented
- In:
  c=++a //a is incremented first then c is assigned
- In the line above a is 5, then c is assigned to a
- Same applies to --

Auto Increment / Decrement

- The following are all equal:
  x=x+1
  x+=1
  x++
- However x++ is not the same as ++x which in C has a meaning.
- x++ means

Precedence Rules

- Operators in an expression are evaluated according to precedence rules

  ( )

  *, /, %

  +, -

  =, +=, *=, /=, -=, %=

  Comma operator

Time For Demo!

```
/*
  Let's try writing some calculations...
 */
#include <stdio.h>

int main() {
  float value1 = 0.0, value2 = 0.0;
  int i1 = 0, i2 = 0;
  /*
    Prompt for values
   */
  printf( "\t\tCalculation Program\n\n" );
  printf( "Please enter two values: " );
  scanf( "%f %f", &value1, &value2 );

  printf( "Their sum: %f\n", value1 + value2 );
  printf( "Their product: %f\n", value1 * value2 );
  printf( "The first minus second: %f\n", value1 - value2 );
  printf( "The second minus first: %f\n",  value2 - value1 );
  printf( "The first divided by the second: %f\n", value1 /
value2 );
  printf( "The second divided by the first: %f\n", value2 /
value1 );

  printf( "Let's try again, as if the values were int\n" );
  i1 = (int) value1;
  i2 = (int) value2;
  printf( "Their sum: %d\n", i1 + i2 );
  printf( "Their product: %d\n", i1 * i2 );
  printf( "The first minus second: %d\n", i1 - i2 );
  printf( "The second minus first: %d\n", i2 - i1 );
  printf( "The first divided by the second: %d\n", i1 / i2 );
  /*
    Just for the record, the modulus operator is only
    defined on int parameters...
   */
  printf( "The modulus of first by second: %d\n", i1 % i2 );
  printf( "The second divided by the first: %d\n", i2 / i1 );
  printf( "The modulus of second by first: %d\n", i2 % i1 );
  return( 0 );
}
```

Summarizing Our Demo!

- Integer Division can result in bugs!
- Types can be forced through conversions
- Operators follow precedence rules
  - parenthesis can change this ordering
  - do use parenthesis to make your intentions clear

Flow of Control

- Like a cook following recipe instructions, computers execute statements one after another
- Certain statements alter this flow of control
  - if
  - if-else
  - While (next unit)
  - do-while (next unit)

Selective Control Flow in C

- Programs often choose between different instructions in a variety of situations
  - sometimes, code must be skipped because it does not apply in the current situation
  - other times, one of several code blocks must be chosen to be executed based on the current situtation

The `if` Statement

- Guarded Action

```
if ( x < y )
{
  printf("x < y");
}
```

The `if` Statement

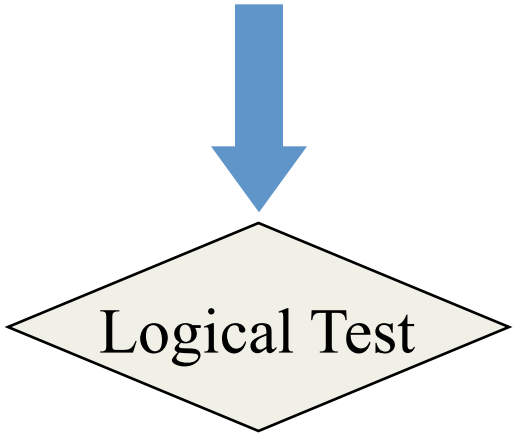
- Guarded Action

```
if ( x < y )
{
  printf("x < y");
}
```

The `if` Statement

- Guarded Action

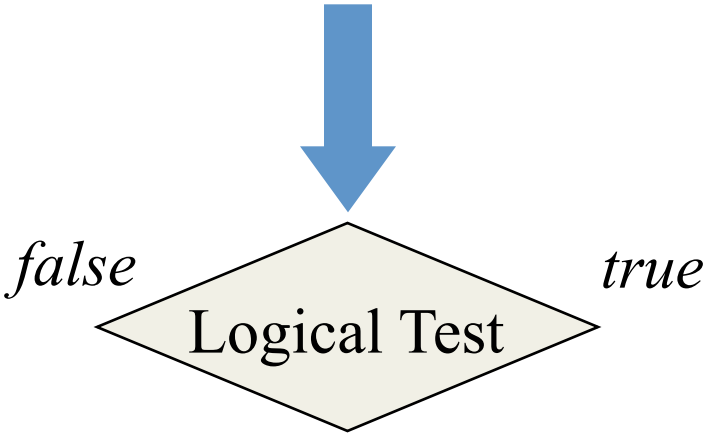
```
if ( x < y )
{
  printf("x < y");
}
```



The `if` Statement

- Guarded Action

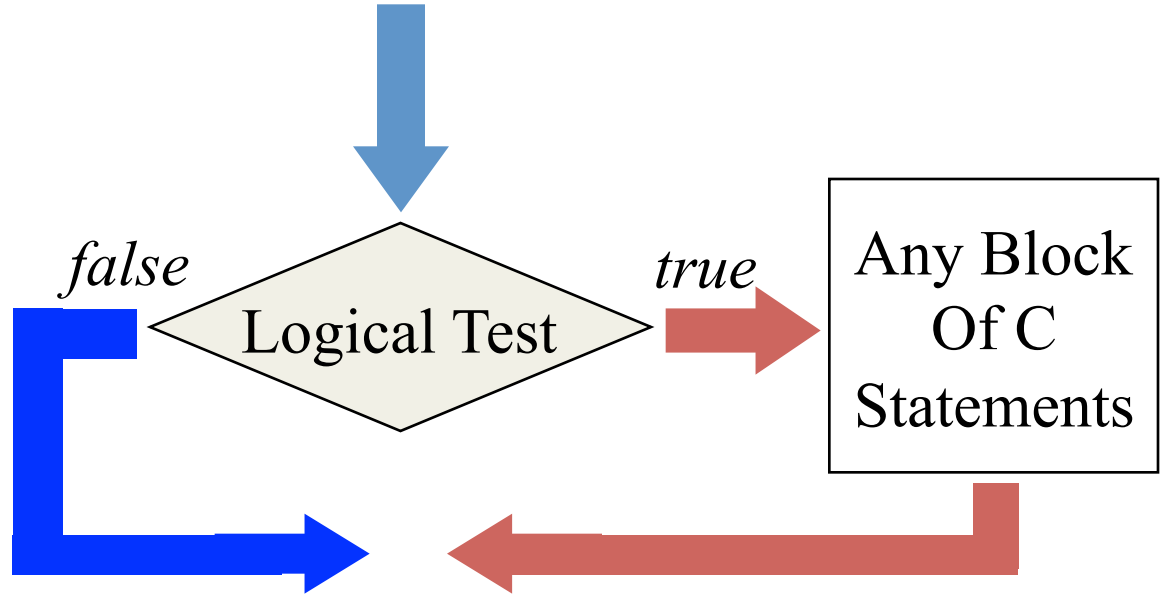
```
if ( x < y )
{
  printf("x < y");
}
```



The `if` Statement

- Guarded Action

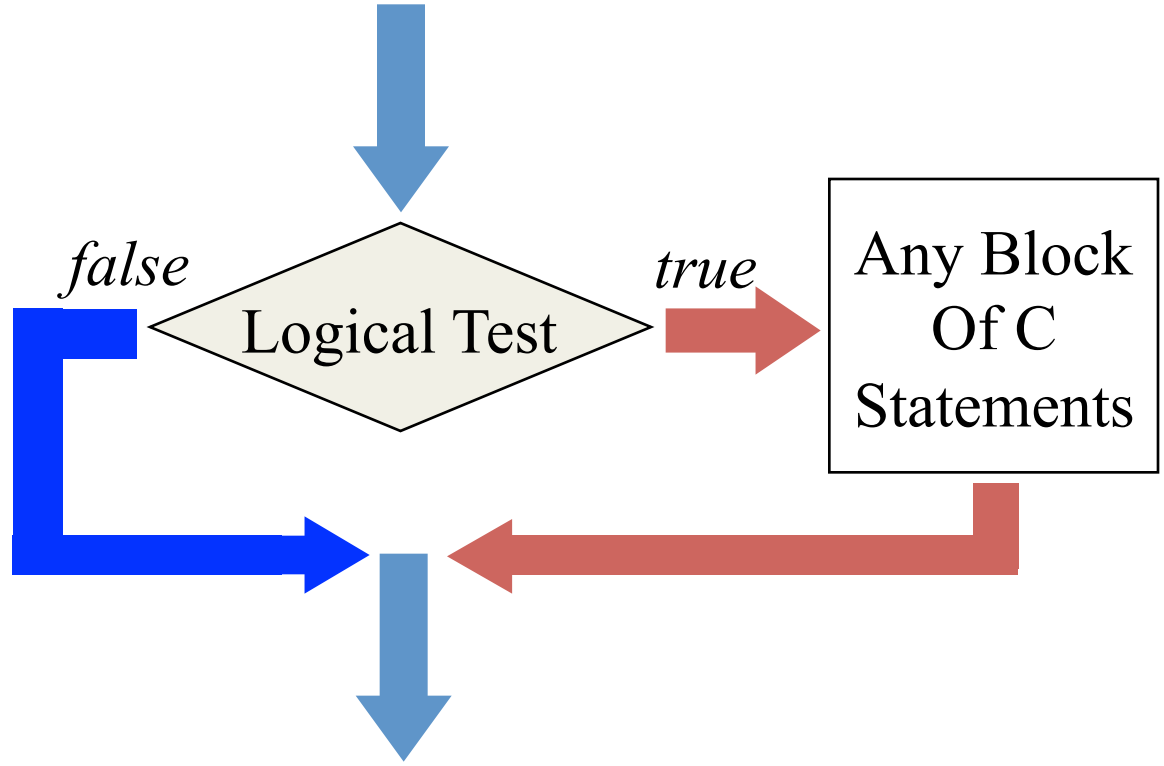
```
if ( x < y )
{
  printf("x < y");
}
```



The `if` Statement

- Guarded Action

```
if ( x < y )
{
  printf("x < y");
}
```



Syntax of if statement

- Recall C is case sensitive
- Start with if
- Condition to be tested comes next between parenthesis
- Condition is usually made of three items:
  - Two operands and a testing operator
- Later we learn about functions – a Boolean function can also be used in lieu of a condition

Comparison Operators

- Operators Testing Ordering

  <, <=, >, >=

- Operators Testing Equality

  ==, !=

Common Mistake

- Assignment (=) is different from test for Equality (==)

```
if (salary = 100000)
{
    printf("You're in!”);
}
```

- Equality is always dangerous when working with real operands

More Complex Expressions

- Examples:

```
if (rate * balance > 1000)
if (a * b != c + d * e)
if (a / b > c)
```

- Add parenthesis to make your intentions clear
- Arithmetic operators have higher precedence than relational operators

  24.00000001 != 24

Logical Operators

- && means AND, || means OR, ! means NOT
  - Please Avoid & and | For Now…
- Truth Tables:

&&	True	False
True	True	False
False	False	False

||	True	False
True	True	True
False	True	False

!	True
True	False
False	True

Logical Operators

- && means AND, || means OR, ! means NOT
  - Please Avoid & and | For Now…
- Truth Tables:

&&	True	False
True	True	False
False	False	False

Both Sides Must Be True For && To Be True…

||	True	False
True	True	True
False	True	False

Both Sides Must Be False For || To Be False…

!	True
True	False
False	True

! Inverts…

Logical Operators

- Examples:
  - true && false =
  - false && true =
  - true || false =
  - false || true =
  - ! true =
  - ! false =

Logical Operators

- Examples:
  - true && false = false
  - false && true = false
  - true || false = true
  - false || true = true
  - ! true = false
  - ! false = true

Logical Operators

- Logical Operators connect expressions
- Examples:

```
if ((0 <= x) && (x > 3))
if ((y != 1) && (x/y > 4))
```

- C uses short-circuit evaluation
  - The evaluation of condition stops because the condition could not possibly be true (in case of &&) or false (in case of ||)

C has no True or False Literals

- Do not compare a Boolean expression to true or false
- 0 is false; everything else is true
- You may create constants true and false but that is a little dangerous

Precedence Rules

- Parentheses
- Unary Operators: +, -, !
- Arithmetic Operators: *, / then +, -, then %
- Comparison Operators: <, <=, >, >=, ==, !=
  then && then ||

Time For Our Selection Demo!

```
/*
  Let's try writing some conditional logic...
 */
#include <stdio.h>

int main() {
  float value1 = 0.0, value2 = 0.0;
  /*
    Prompt for values
   */
  printf( "\t\tConditional Logic Program\n\n" );
  printf( "Please enter two values: " );
  scanf( "%f %f", &value1, &value2 );

  if (value1 < value2) {
          printf( "value1 is less than value2\n" );
  }
  if (value1 > value2) {
          printf( "value1 is greater than value2\n" );
  }
  if (value1 <= value2) {
          printf( "value1 is less than or equal value2\n" );
  }
  if (value1 >= value2) {
          printf( "value1 is greater than or equal value2\n" );
  }
  if (value1 == value2) {
          printf( "value1 equals value2\n" );
  }
  if (value1 != value2) {
          printf( "value1 does not equals value2\n" );
  }
  /*
    Just for the record, due to rounding errors, it
        is very dangerous to test for equality on floating
        point numbers
   */
  return( 0 );
}
```

Summarizing Our Second Demo!

- Proper indentation helps express your intentions
  - But remember, the compiler ignores whitespace....

The `if-else` Statement

- Alternative Action

```
if ( x < y )
{
  x++;
}
else
{
  y++;
}
```

The `if-else` Statement

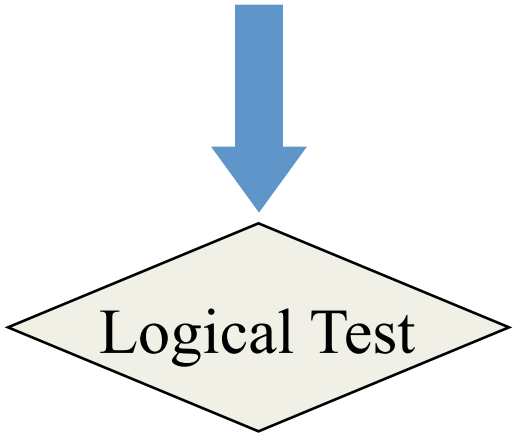
- Alternative Action

```
if ( x < y )
{
  x++;
}
else
{
  y++;
}
```

The `if-else` Statement

- Alternative Action

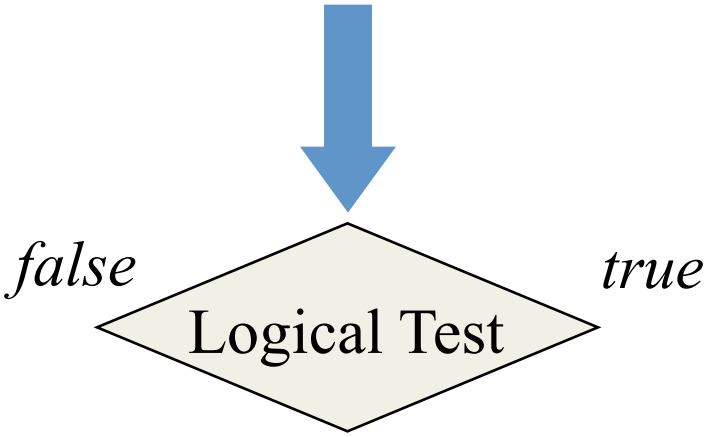
```
if ( x < y )
{
  x++;
}
else
{
  y++;
}
```



The `if-else` Statement

- Alternative Action

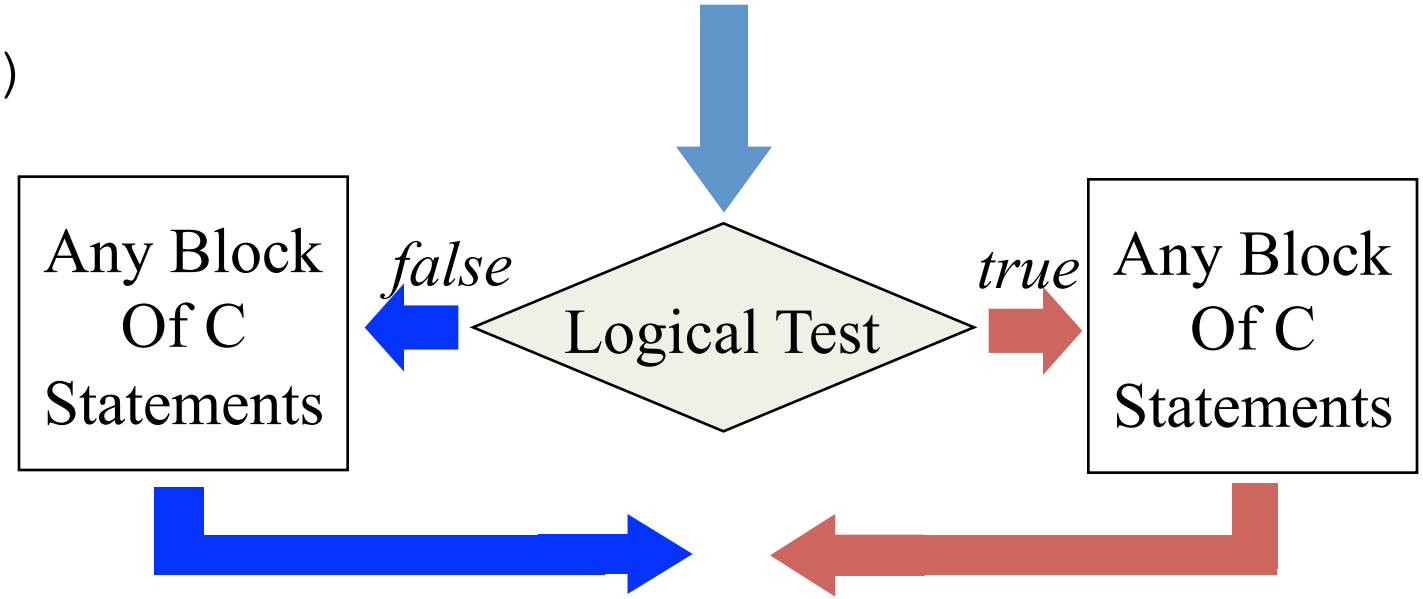
```
if ( x < y )
{
  x++;
}
else
{
  y++;
}
```



The `if-else` Statement

- Alternative Action

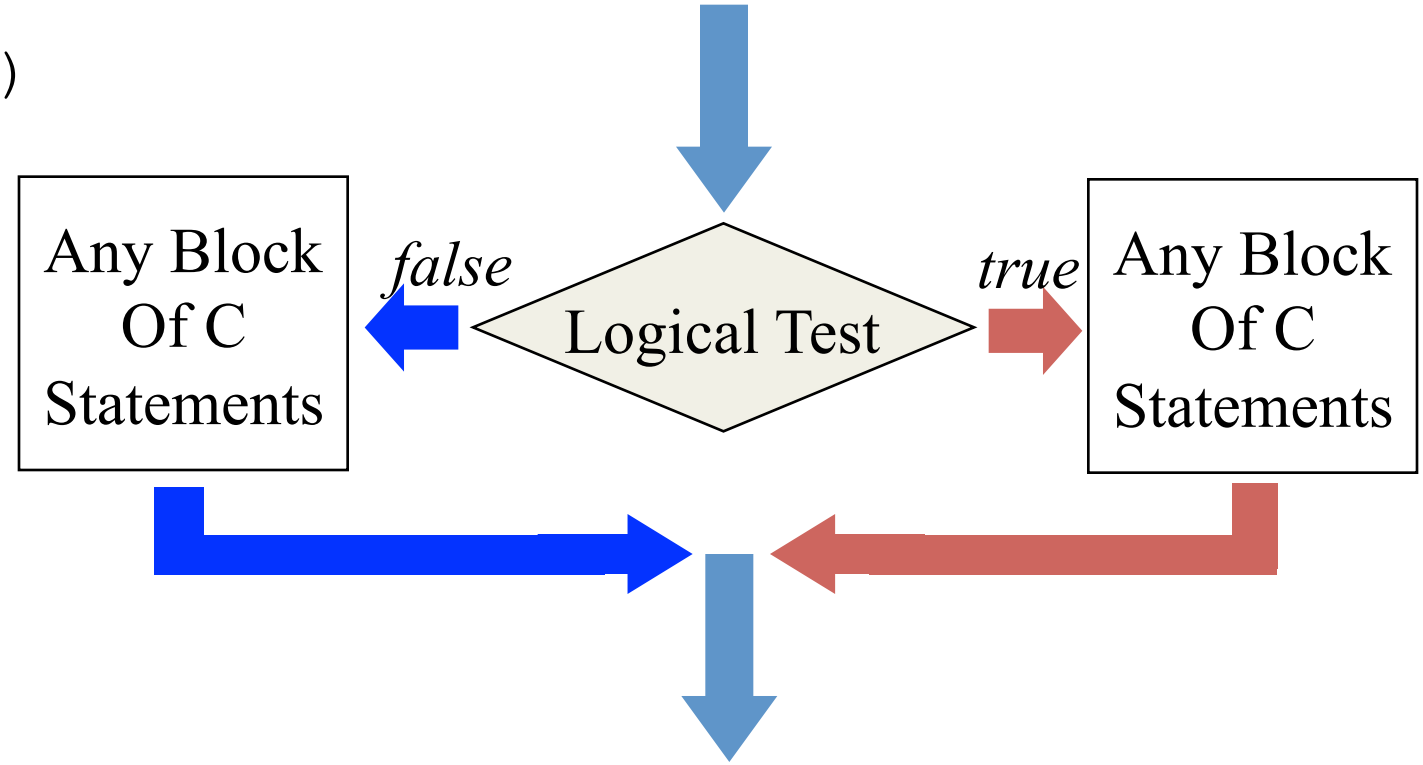
```
if ( x < y )
{
   x++;
}
else
{
   y++;
}
```



The `if-else` Statement

- Alternative Action

```
if ( x < y )
{
   x++;
}
else
{
   y++;
}
```



Nested Conditional Statements

- Selection Statements can be used in combination
- Just be sure that the `else` clause is not dangling…

```
if (precipitating)
 if (temperature < 32)
   printf("It's snowing");
else // HMMM...
 printf( "It's raining" );
```

Time For Nested Selection Demo!

```
/*
  Demo of nested conditional statements...
 */
#include <stdio.h>

int main() {
  int temperature;
  /*
    Prompt for values
   */
  printf( "\t\tNested Logic Program\n\n" );
  printf( "Please enter today's temperature: " );
  scanf( "%d", &temperature );

  if (temperature < 50) {
        printf( "Gosh, it feels cold...\n" );
        if (temperature < 32) {
              printf( "And it looks like it's freezing...\n" );
        }
        else if (temperature < 40) {
              printf( "And it's nearly freezing...\n" );
        }
        else {
              printf( "But atleast it's not freezing cold!\n" );
        }
  }
  else if (temperature > 90) {
        printf( "Gosh, it's hot...\n" );
        if (temperature > 110) {
              printf( "And it's just boiling... head for air
conditioning...\n" );
        }
        else if (temperature > 100) {
              printf( "Atleast it's not boiling...\n" );
        }
        else {
              printf( "What a heat wave!!\n" );
        }
  }
  else {
        printf( "Doesn't California have a nice climate!\n" );
  }
  return( 0 );
}
```

Summarizing Our Third Demo!

- Nested conditionals make for complex scenarios
- Use parentheses to prevent A dangling `else`
- Remember only one guarded action or alternative is chosen

Selective Control Flow in C

- Programs often choose between different instructions in a variety of situations
  - sometimes, code must be skipped because it does not apply in the current situation
  - other times, one of several code blocks must be chosen to be executed based on the current situtation

So far we did

- If statament
- If – else statement
- Next we do if – elseif statement

The `if-else if-else` Statement

- Multiple Action

```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```

The `if-else if-else` Statement

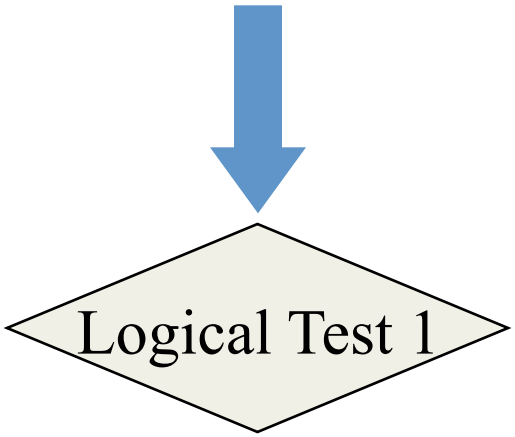
- Multiple Action

```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```

The `if-else if-else` Statement

- Multiple Action

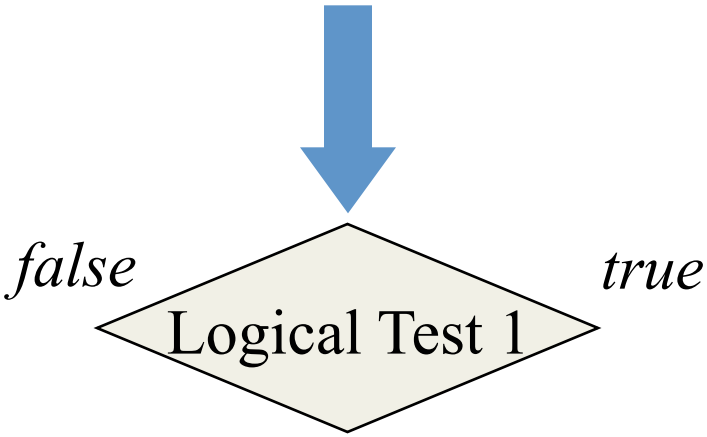
```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Multiple Action

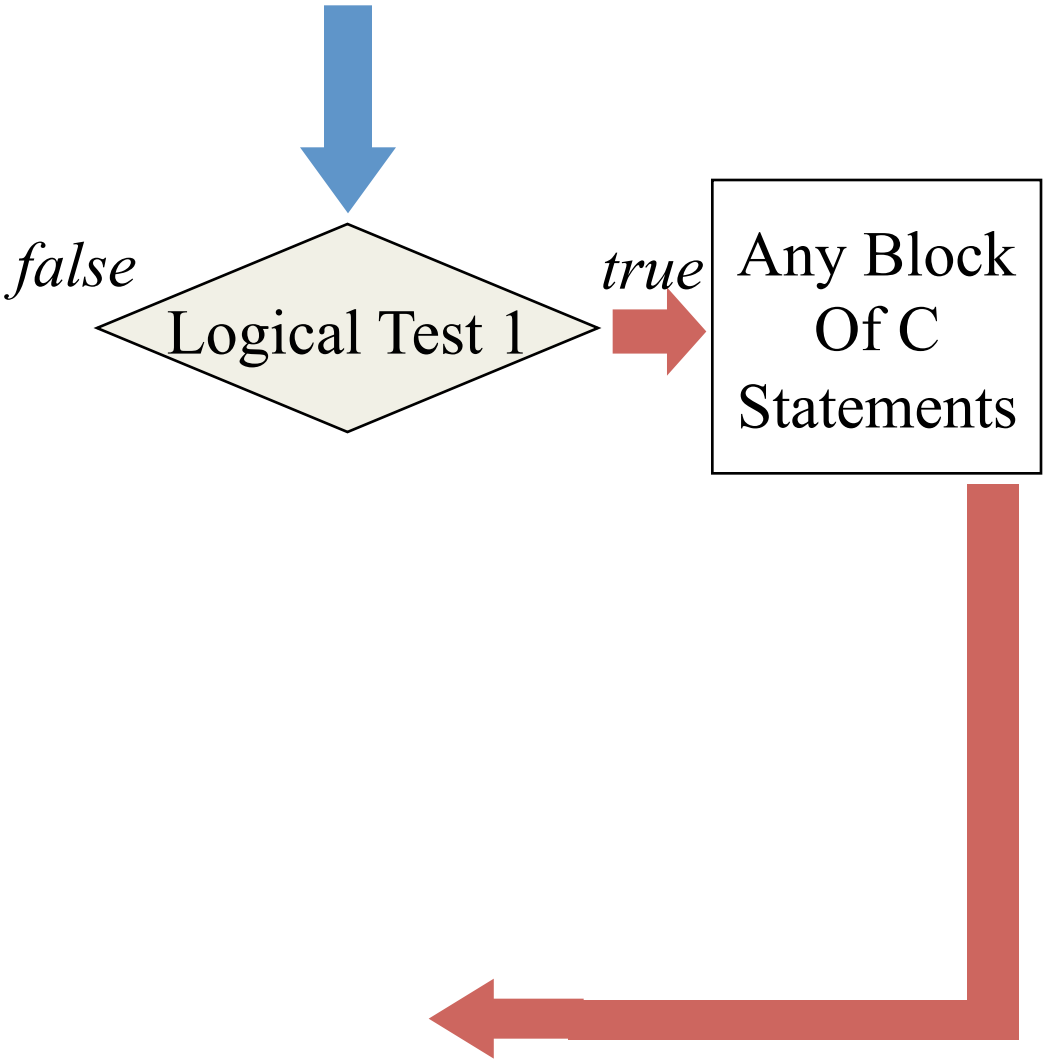
```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Multiple Action

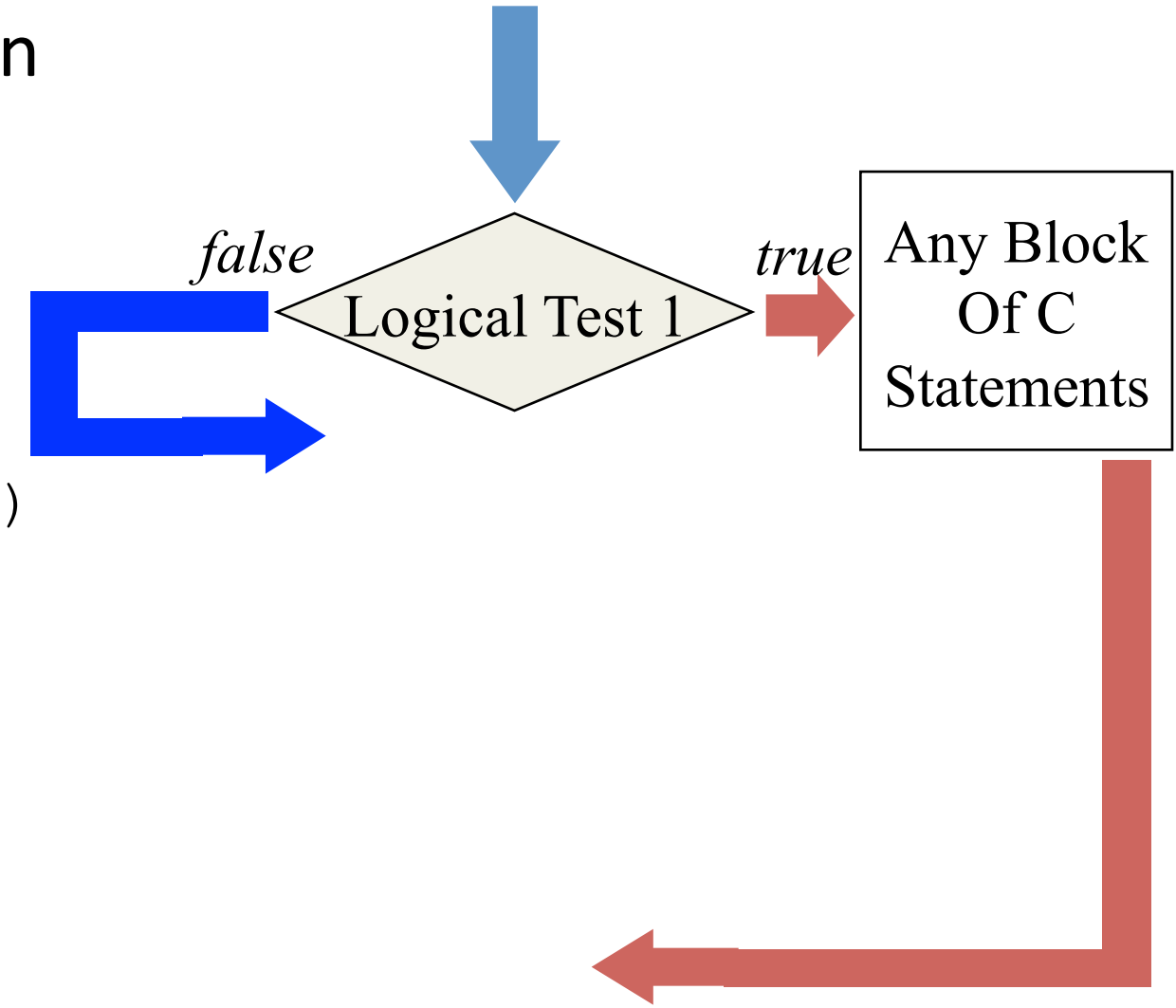
```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Multiple Action

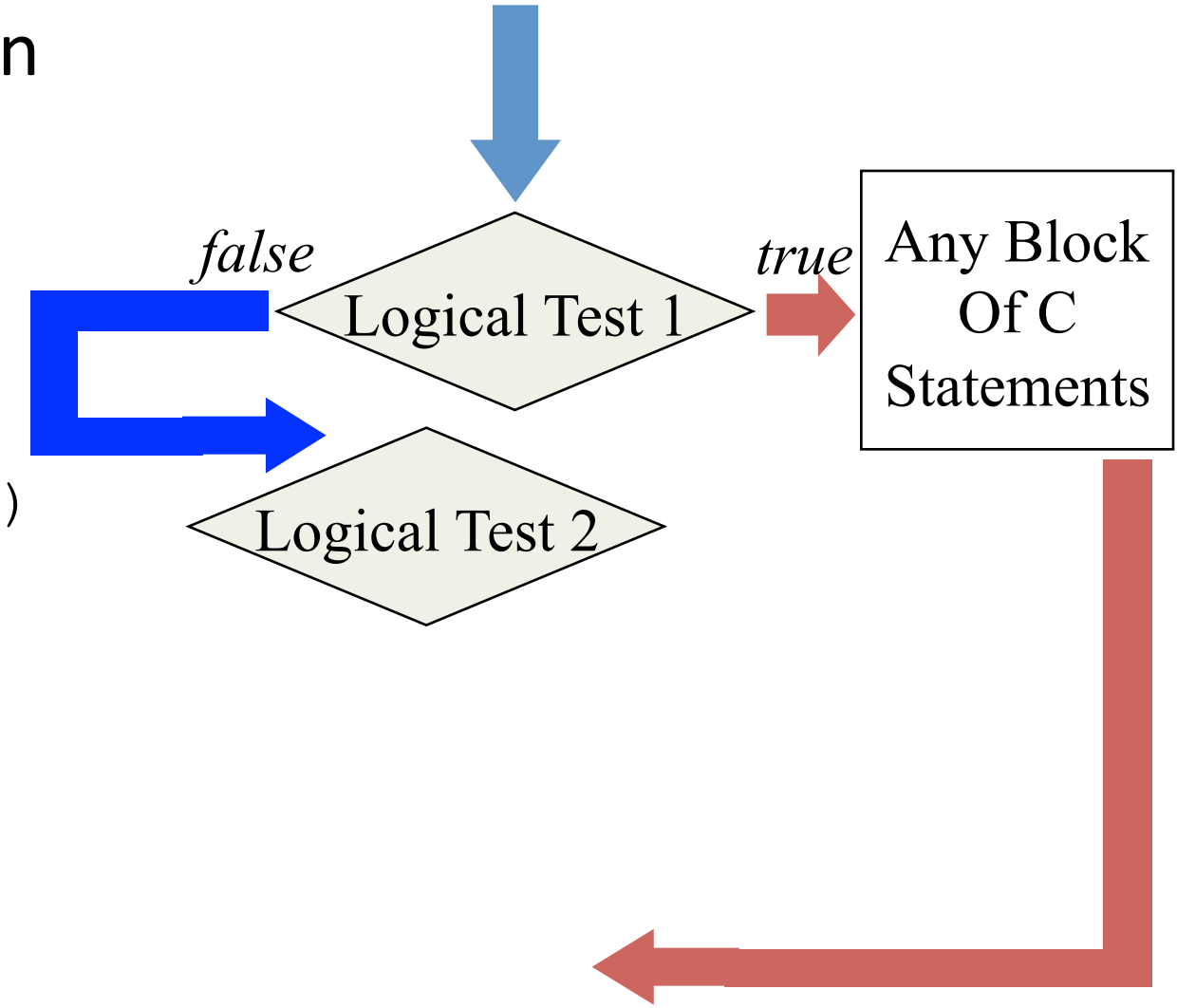
```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Multiple Action

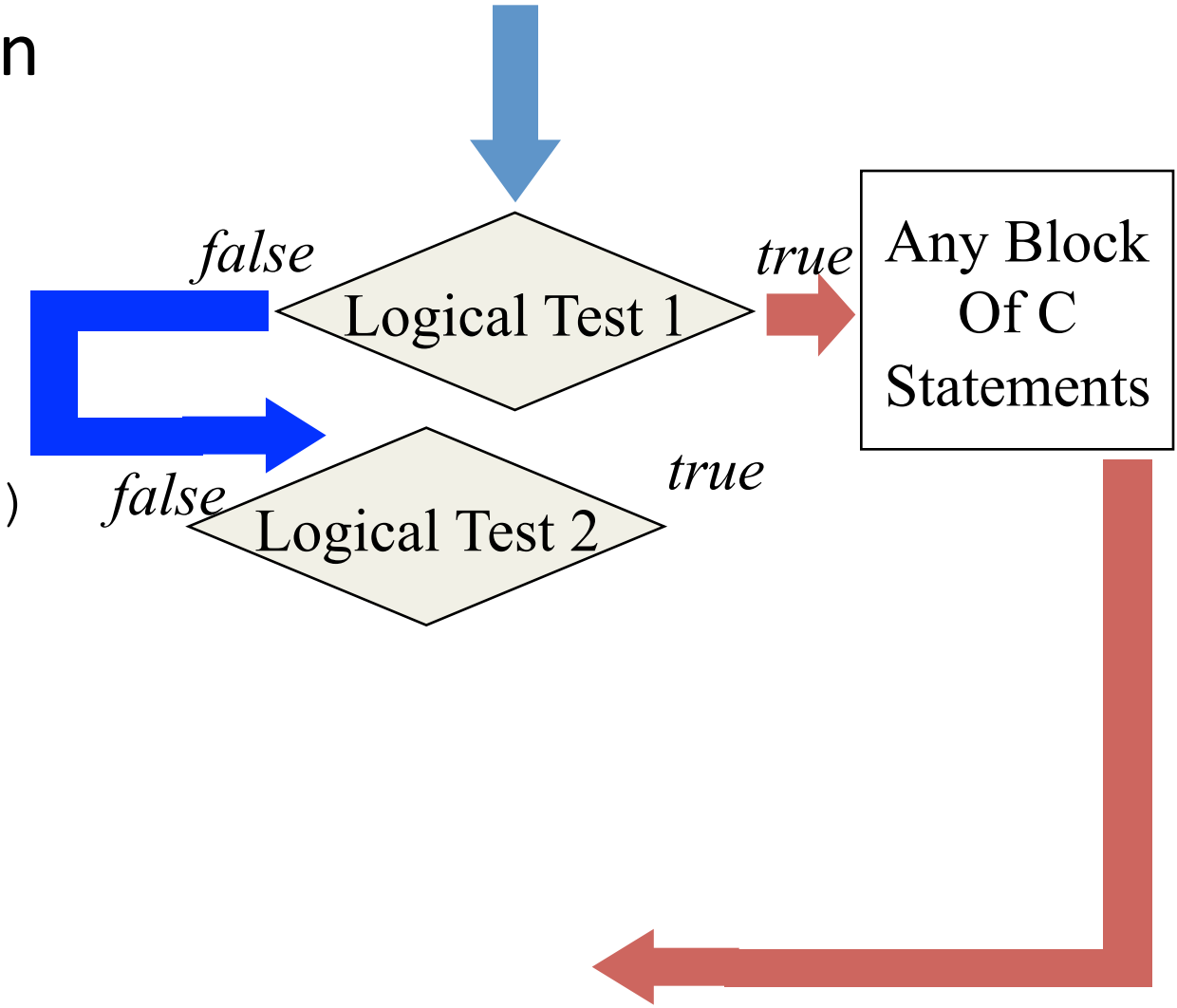
```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Multiple Action

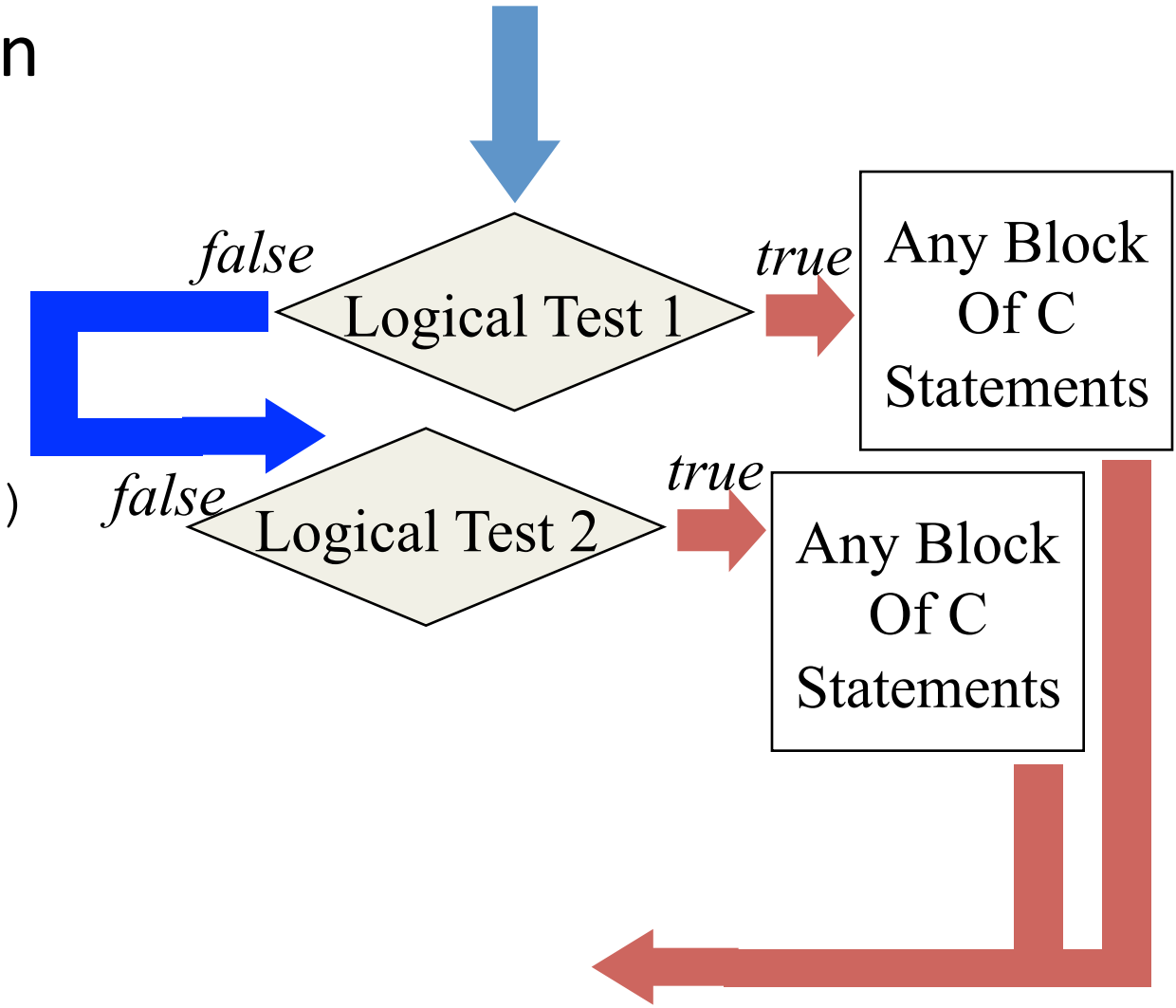
```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Multiple Action

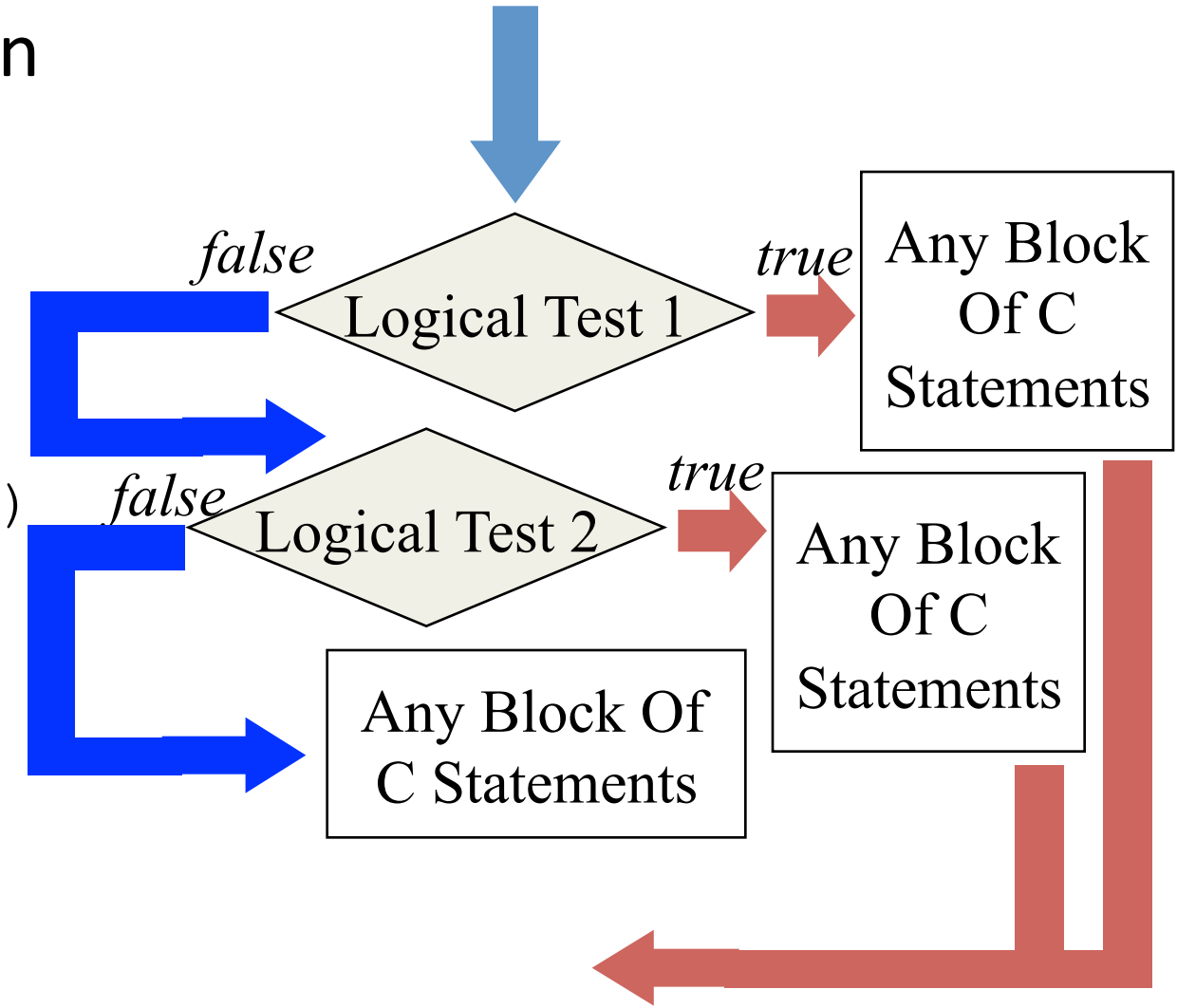
```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Multiple Action

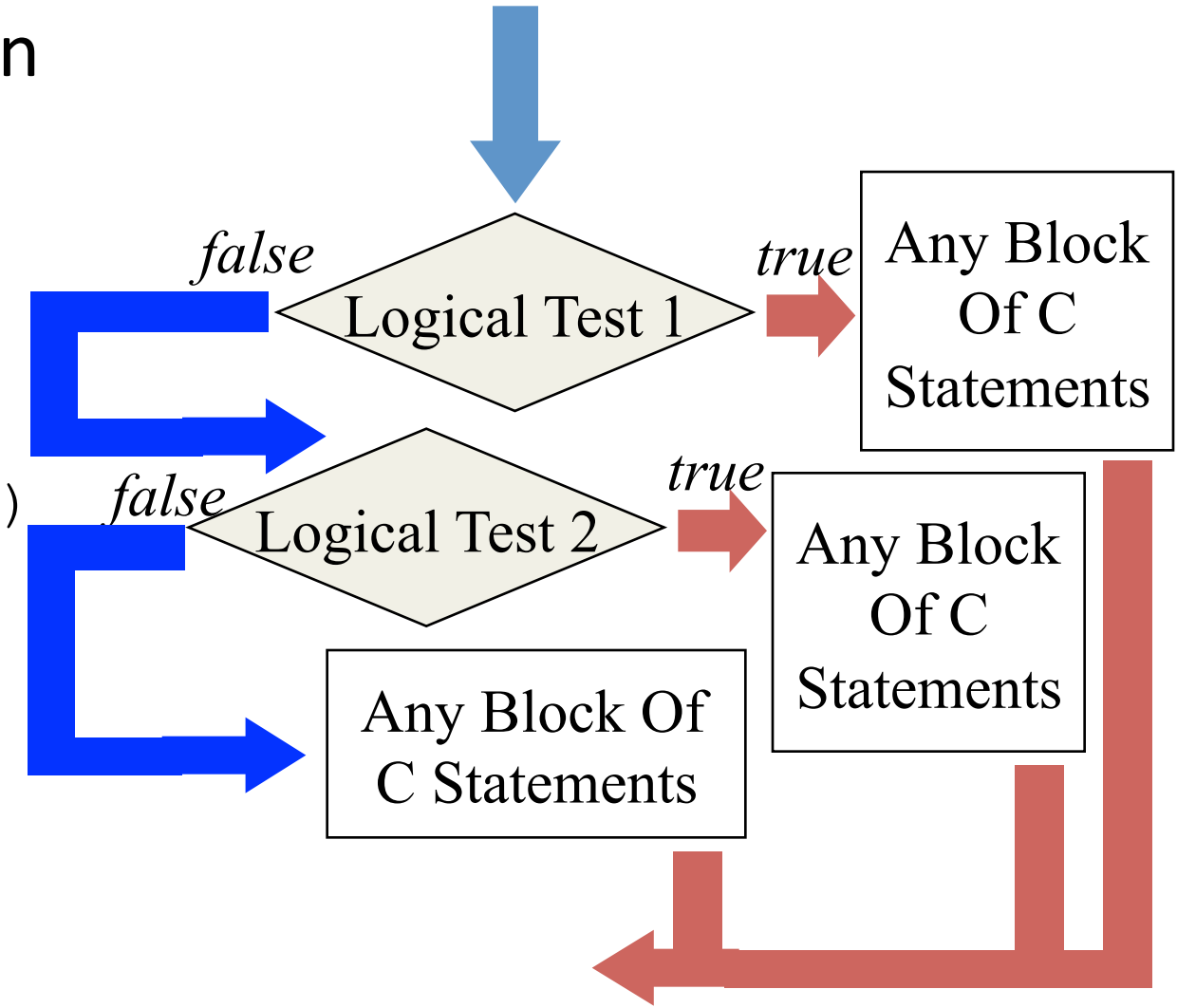
```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Multiple Action

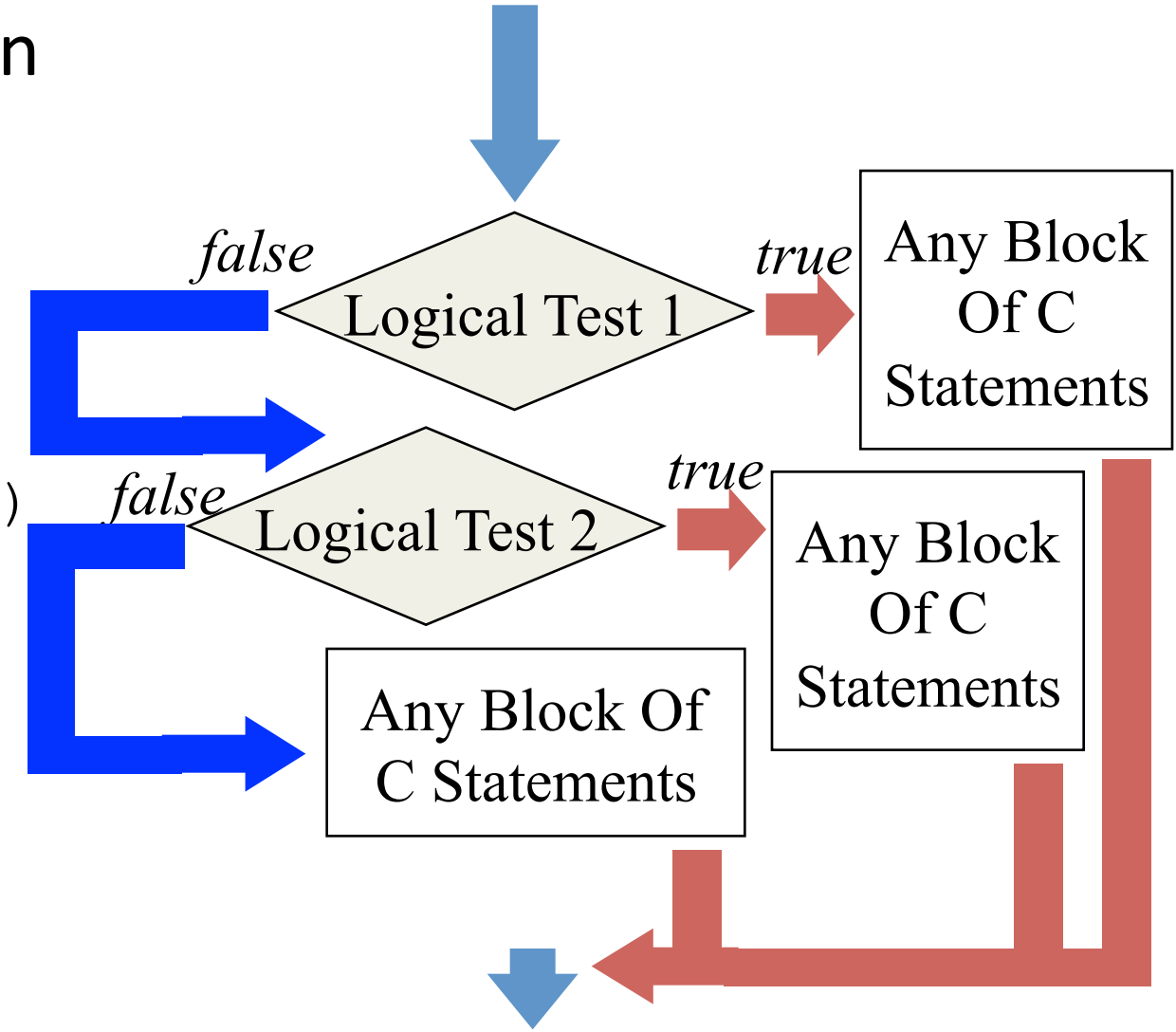
```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Multiple Action

```
if ( x < y )
{
  x++;
}
else if ( x > y )
{
  y++;
}
else {
  x++; y++;
}
```



The `if-else if-else` Statement

- Any Number Of `else-if` Alternatives Is Allowed
- The `else` Clause Is Completely Optional

The `switch` Statement

- An Alternative To Lots Of `else-if` Choices

```
switch( option ) {
  case 1:
     printf( “1” );
     break;
  case 2:
     printf( “2” );
     break;
  default:
     printf( “other” );
}
```

The `switch` Statement

- An Alternative To Lots Of `else-if` Choices

switch expression must evaluate to an integral value

```
switch( option ) {
  case 1:
     printf( “1” );
     break;
  case 2:
     printf( “2” );
     break;
  default:
     printf( “other” );
}
```

The `switch` Statement

- An Alternative To Lots Of `else-if` Choices

switch expression must evaluate to an integral value

choice must be a constant value

```
switch( option ) {
  case 1:
     printf( “1” );
     break;
  case 2:
     printf( “2” );
     break;
  default:
     printf( “other” );
}
```

The `switch` Statement

- An Alternative To Lots Of `else-if` Choices

switch expression must evaluate to an integral value

choice must be a constant value

break exits this control structure

```
switch( option ) {
  case 1:
     printf( “1” );
     break;
  case 2:
     printf( “2” );
     break;
  default:
     printf( “other” );
}
```

The `switch` Statement

- An Alternative To Lots Of `else-if` Choices

```
switch( option ) {
  case 1:
     printf( “1” );
     break;
  case 2:
     printf( “2” );
     break;
  default:
     printf( “other” );
}
```

switch expression must evaluate to an integral value

choice must be a constant value

break exits this control structure

default case for when no matches occur; completely optional

Time For Our Next Demo!

```
/*
  Let's try writing a switch statement...
 */
#include <stdio.h>

int main() {
  char letter;
  /*
    Prompt for values
   */
  printf( "\t\tCase Statement Program\n\n" );
  printf( "Please enter a letter to inspect: " );
  scanf( "%c", &letter );

  /*
    Just for the record, you can only switch on a
          integral value.  The char datatype is just another
          name for the set of ints between 0 and 255, so you
          can switch on chars or ints
   */
  switch( letter ) {
    /*
            Individual letters must be single-quoted. Individual letters
map directly to constant integer values based on the ASCII table
which we will learn about in upcoming units.  The value of each
case must be a constant value, not an expression or variable.  This
often makes switch statements not applicable to your situation.
 */
          case 'a':
          case 'e':
          case 'i':
          case 'o':
          case 'u':
          case 'y':
 /*
Lacking break statements in the upper listed cases, they will
 all "fall thru"to the set of statements shown here.

          While at first this may seem very convenient, this is actually
the number one programming bug worldwide.  Namely, that folks
forget that all the above cases are collapsing down to the code
shown below.  So use this form with great caution, as it often leads
to bugs...
                 */
                printf( "a nice lowercase vowel!\n" );
                break;
        case 'A':
        case 'E':
        case 'I':
        case 'O':
        case 'U':
        case 'Y':
                printf( "a nice UPPERCASE vowel!\n" );
                break;
        case '0':
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
        case '6':
        case '7':
        case '8':
        case '9':
                printf( "a nice number!\n" );
                break;
        default:
                /*
                  The default case is the one selected when
        no other cases actually match the switched data
                 */
                printf( "this is not something I recognize...\n" );
                break;
  }
  /*
    Just for the record, due to rounding errors, it   is very dangerous
to test for equality on floating  point numbers
  */
  return( 0 );
}
```

Summarizing Switch Demo!

- Pick the control flow that most naturally fits your intentions
- Without a `break`, `switch` will continue executing next `case`
- `Break` statement exits any loop construct
- Remember only one alternative is chosen

Another example of `switch`

- Assume x is an integer and we want to see if it is even or not.

```
// if x is even, then x mod 2 is 0.
Switch(n)    {
}
```

In order to check if x is even,
x mod 2 should be 0.
Otherwise, x is not even.
Hence, n should be the expression:
x % 2.
You can use n=x%2 or place x%2 here

Another example of `switch`

- Assume x is an integer and we want to see if it is even or not.

```
// if x is even, then x mod 2 is 0.
Switch(x%2) {
  case 0:


}
```

There are 2 possibilities:

x %2 is 0 means x is even

Another example of `switch`

- Assume x is an integer and we want to see if it is even or not.

```
// if x is even, then x mod 2 is 0.
Switch(x%2) {
  case 0:
     printf("%d is even",x);
     break;
.
.
.
}
```

must break so next case is not checked.

Another example of `switch`

- Assume x is an integer and we want to see if it is even or not.

```
// if x is even, then x mod 2 is 0.
Switch(x%2) {
  case 0:
     printf("%d is even",x);
     break;
default:
     …
}
```

Anything else means x is not even

Another example of `switch`

- Assume x is an integer and we want to see if it is even or not.

```
// if x is even, then x mod 2 is 0.
Switch(x%2) {
  case 0:
     printf("%d is even",x);
     break;
default:
     printf( "other" );
}
```

Other examples of `switch`

- If x is any of 2,4 the output must be “ok’, and if x is any of 3,5 the output must be “not ok”, and any other number the output must be “invalid”.

Other examples of `switch`

- If x is any of 2,4 the output must be “ok’, and if x is any of 3,5 the output must be “not ok”, and any other number the output must be “invalid” .
- **<u>Long solution:</u>**

```
switch (x)  {
 case 2:
     printf(“ok”)
     break;
 case 4:
    printf(“ok);
    break;
case 3:
    printf(“not ok);
    break;
case 5:
    printf(“not ok”);
    break;
default:
    printf(“invalid”);
}
```

Other examples of `switch`

- If x is any of 2,4 the output must be “ok’, and if x is any of 3,5 the output must be “not ok”, and any other number the output must be “invalid” .
- **<u>Short solution:</u>**

```
switch (x)  {
 case 2:
 case 4:
    printf(“ok);
    break;
case 3:
case 5:
  printf(“not ok”);
  break;
default:
  printf(“invalid”);
}
```

Other examples of `switch`

- If x is any of 2,4 the output must be “ok’, and if x is any of 3,5 the output must be “not ok”, and any other number the output must be “invalid” .

Short solution:

```
switch (x)  {
 case 2:
 case 4:
    printf(“ok);
    break;
case 3:
case 5:
  printf(“not ok”);
  break;
default:
  printf(“invalid”);
}
```

In this case, there is no break;

Other examples of `switch`

- Assume x can be any of 2,4 the output must be “ok’, and if x is any of 3,5 the output must be “not ok”, and any other number the output must be “invalid”.

Short solution:

```
switch (x)  {
 case 2:
 case 4:
    printf(“ok);
    break;
case 3:
case 5:
  printf(“not ok”);
  break;
default:
  printf(“invalid”);
}
```

No break in case 2, and next case is also executed.

Other examples of `switch`

- Assume x can be any of 2,4 the output must be “ok’, and if x is any of 3,5 the output must be “not ok”, and any other number the output must be “invalid”.

Short solution:

```
switch (x)  {
 case 2:
 case 4:
    printf(“ok);
    break;
case 3:
case 5:
  printf(“not ok”);
  break;
default:
  printf(“invalid”);
}
```

This is called Fallout.
A comment is recommended

Other examples of `switch`

- Assume x can be any of 2,4 the output must be “ok’, and if x is any of 3,5 the output must be “not ok”, and any other number the output must be “invalid”.

Short solution:

```
switch (x)  {
 case 2: //fallout
 case 4:
    printf(“ok);
    break;
case 3:
case 5:
  printf(“not ok”);
  break;
default:
  printf(“invalid”);
}
```

This is called Fallout.
A comment is recommended

Summary

- Expressions
- Various kinds of selective flow of control
  - if and all its variations
  - switch – break