CIS 22A
Review: Selection – Making Decisions

Logical expression − an expression that can be either **true** or **false**. Logical expressions are created using relational and/or logical operators.

Relational Operators: `<` `<=` `>` `>=` `==` `!=`

Logical Operators:

not	
false	true
true	false

and	false	true
false	false	false
true	false	true

or	false	true
false	false	true
true	true	true

Evaluation of logical expressions

–	complement of `==`	is	`!=`	`!(a == b)`	is	`a != b`
–	complement of `<`	is	`>=`	`!(a < b)`	is	`a >= b`
–	complement of `>`	is	`<=`	`!(a > b)`	is	`a <= b`
–	complement of `&&`	is	`||`	`!(a && b)`	is	`!a || !b`
	(De Morgan's Rule)			`!(a || b)`	is	`!a && !b`

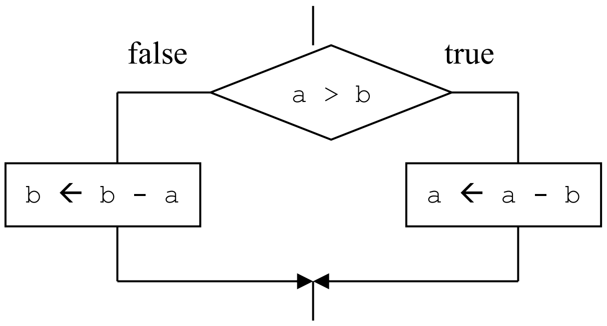
Precedence/ Associativity	Arithmetical Operators	Relational Operators	Logical Operators	Assignment Operators
15 *Right to left*	`+` *plus* `-` *minus*		`!`	
14 *Left to right*	`*` `/` `%`			
13 *Left to right*	`+` `-`			
10 *Left to right*		`<` `<=` `>` `>=`		
9 *Left to right*		`==` `!=`		
5 *Left to right*			`&&`	
4 *Left to right*			`||`	
2 *Right to left*				`=` `+=` `-=` `*=` `/=` `%=`

Expressions connected by `&&` and/or `||` are evaluated left to right, and it is guaranteed that the evaluation will stop as soon as the truth or falsehood is known.

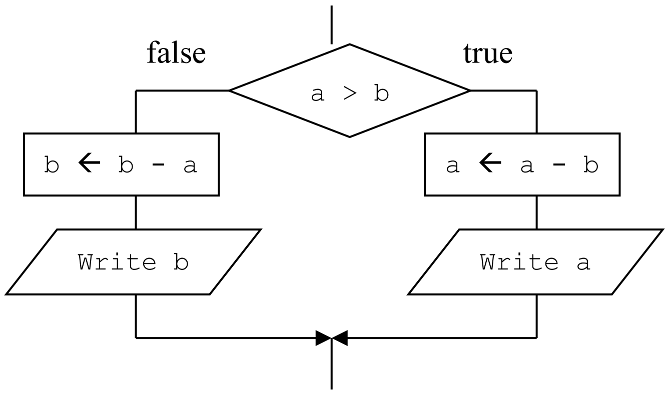
```
n != 0 && a / n > 10
```

// when n is 0, `a / n > 10` is not evaluated, because `false` and anything is `false`

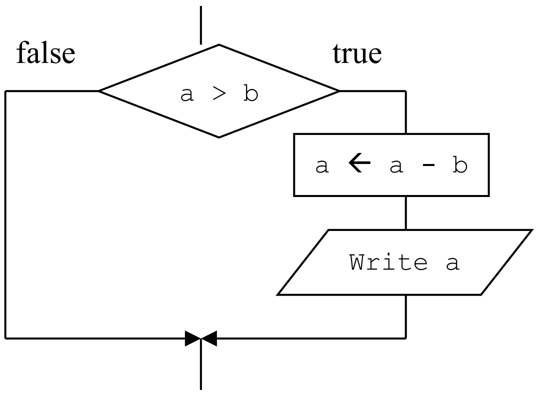
Two-Way Selection – a logical expression is evaluated; if it is true, one or more actions is/are executed, if it is false, another action or group of actions is executed.



```
if( a > b )
    a = a - b;
else
    b = b - a;
```

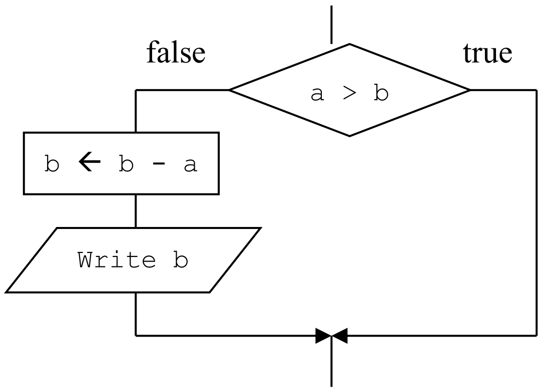


```
if( a > b )
{
   a = a - b;
   cout << a;
}
else
{
   b = b - a;
   cout << b;
}
```



```
if( a > b )
{
   a = a - b;
   cout << a;
}
else // else not needed!
   ;

if( a > b )
{
   a = a - b;
   cout << a;
}
```

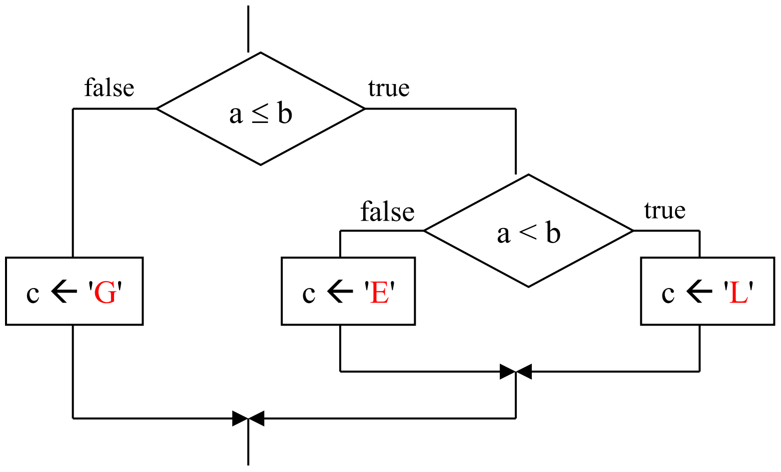


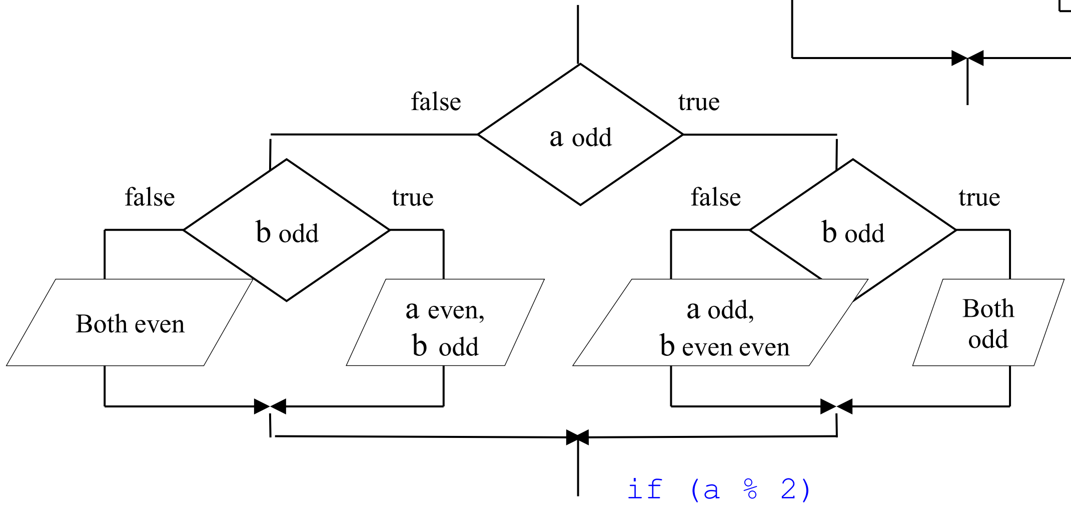
```
// ugly
if( a > b )
    ;
else
{
   a = a - b;
   cout << a;
}

// recommended
if( a <= b )
{
   a = a - b;
   cout << a;
}
```

Nested if Statements

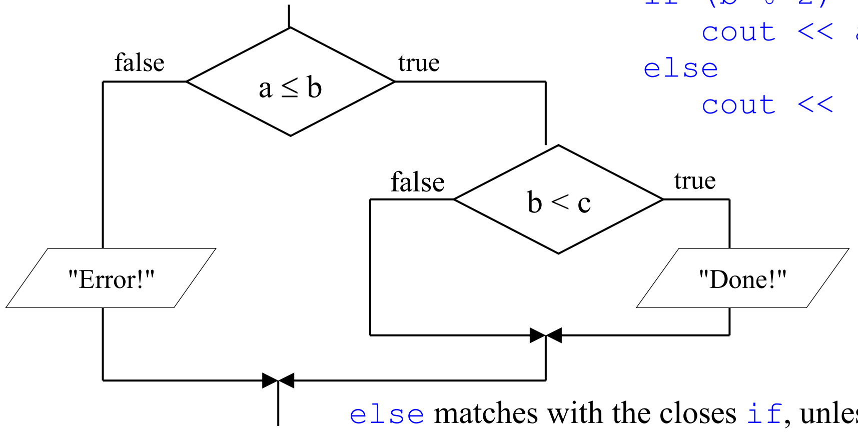
```
if (a <= b)
   if (a < b)
      c = 'L';
   else
      c = 'E';
else
   c = 'G'
```





```
if (a % 2)
   if (b % 2)
      cout << "Both odd\n";
   else
      cout << a << " odd " << b << " even\n";
else
   if (b % 2)
      cout << a << " even " << b << " odd\n";
   else
      cout << "Both even\n";
```

The Dangling else Problem



`else` matches with the closes `if`, unless either add `;` (empty statement, meaning "do nothing") or use { } as shown below:

```
if (a <= b)
   if (b < c)
      cout << "Done!\n";
   else
      ;
else
    cout << "Error!\n";
```

```
if (a <= b)
{
   if (b < c)
      cout << "Done!\n";
}
else
    cout << "Error!\n";
```

Multi -Way Selection – choose among several options

else if – is used to enhance the readability of the code; it is to be used when the same variable is being compared in all tests with different constant values.

```
if (color == 'B')
    cout << "Strong";
else
    if (color == 'G')
        cout << "Growth";
    else
        if (color == 'R')
            cout << "Love";
        else
            if (color == 'Y')
                cout << "Happy";
```

```
if (color == 'B')
    cout << "Strong";
else if (color == 'G')
    cout << "Growth";
else if (color == 'R')
    cout << "Love";
else if (color == 'Y')
    cout << "Happy";
```

switch – it is to be used when the same integral expression is being compared using the equal sign with different constant values.

```
switch (color)
{
   case 'B': cout << "Strong";
             break;
   case 'G': cout << "Growth";
             break;
   case 'R': cout << "Love";
             break;
   case 'Y': cout << "Happy";
             break;
}//  end of switch
```

`break` – skips at the first statements after the switch; once that it is decided where to start based on the constant value, statements are executed sequentially until a break or the end of the switch is encountered.

```
switch (op)
{
   case '+': sum = a + b;
             cout << sum << endl;
             break;
   case '/': quotient = a / b;
             cout << quotient
                  << endl;
   case '%': rem = a % b;
             cout << rem << endl;
             break;
   default:  cout << "Error" << endl;
             break;
}//  end of switch
```

`{}` – are mandatory for the switch statement only, not for each case.

`default` – may be omitted; it is executed when the switch selector's value does not have a match among the case constants.

Conditional operator – a ternary operator: it requires three operands; it provides an alternative way to write if-else. **? :**

```
if( a > b )
   max = a;                    max = (a > b) ? a : b;
else
   max = b;
```

Conditional expression – an expression created using the conditional operator

expression1 **?** expression2 **:** expression3

Character Conversion Functions – one parameter, the character to be converted, and as returned value, the converted character

`toupper`	// if lowercase, returns the uppercase, otherwise returns the same
`tolower`	// if uppercase, returns the lowercase, otherwise returns the same

```
#include <cctype>

//. . .
    cin >> option;
    option = toupper(option);
```