CSCE 121

Introduction to Program Design & Concepts

Stuff You Already Know, Only in C++ Now

Dr. Tim McGuire

Grateful acknowledgment to Dr. Philip Ritchey and Dr. Michael Moore for some of the material on which these slides are based.

Variables

Some Definitions

- **Object:** region of memory with a type that specifies the representation of the data stored there and what operations can be performed on it
- **Variable:** object with a name
- **Value:** data that may be stored in a variable
- **Declaration:** statement that gives a name to an object
- **Definition:** declaration that also sets aside memory for a variable
  - **Assignment**: statement that gives a value to an object

Names / Identifiers

- For variables, functions, types, …
- Rules:
  - Start with a letter or underscore
  - Only composed of letters, digits and underscores (_)
  - Cannot use keywords (e.g. int, if, while, double)

Variables

- Identifier is associated with a specific location in memory (i.e. address).
- Use them in programs as if they were the value.
- In the background, the compiler sets things up to dereference the variable identifier (i.e. get the value held in the address).
- The variable type dictates how the bits will be interpreted. (More on types and data representation later…)

Declaration, Definition, & Initialization

- Declare
  - Say what an identifier is and what type of object it refers to.
  - Connects a name to an object.
- Define
  - Sets aside memory for the variable.
- Initialize
  - Assign value to variable for the first time.

- Note: the zyBook conflates Declaration and Definition, but they are technically different.

Examples

- int z; (declaration and definition)
- extern int z; (declaration)
  - This is rare for variables of base types. (i.e. we won't do this…)
- int z = 7; (declaration, definition, and initialization)

Control Structures

Structured Programming

- Programmers used to use ‘goto’. This is universally considered BAD PRACTICE.
  - Edgar Dijkstra: Go To Statement Considered Harmful
  - https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf
- This resulted in “Spaghetti code” that was hard to follow / understand / debug.
- Structured programming saw all programs as composed of three control structures.
- While not the only programming paradigm, aspects of structured programming still apply today.

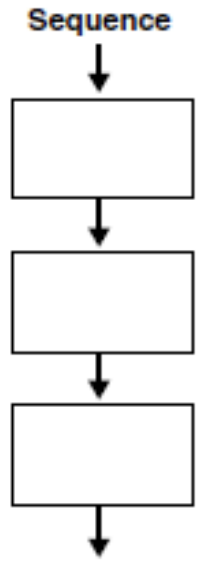
Control Structures

- Sequence
- Selection
- Repetition

Control Structures

- **Sequence**
- Selection
- Repetition

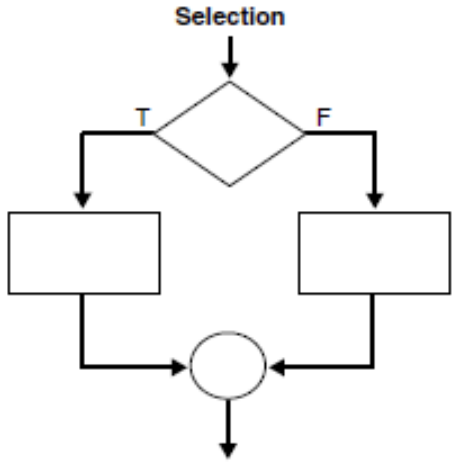
- The ordering of statements
- Can include function calls
- Essentially no “Decisions” are made.



Control Structures

- Sequence
- **Selection**
- Repetition

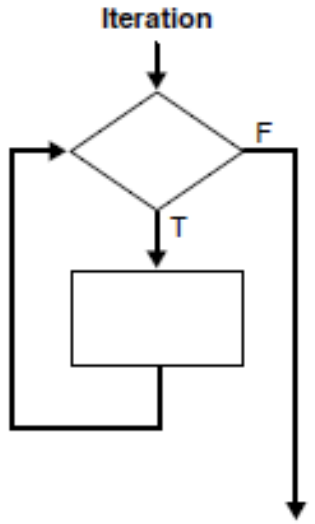
- Also called “branching”
- Allows a block of code to be executed or not based on a “Decision/Question”
- Allows for different paths through the code



Control Structures

- Sequence
- Selection
- Iteration

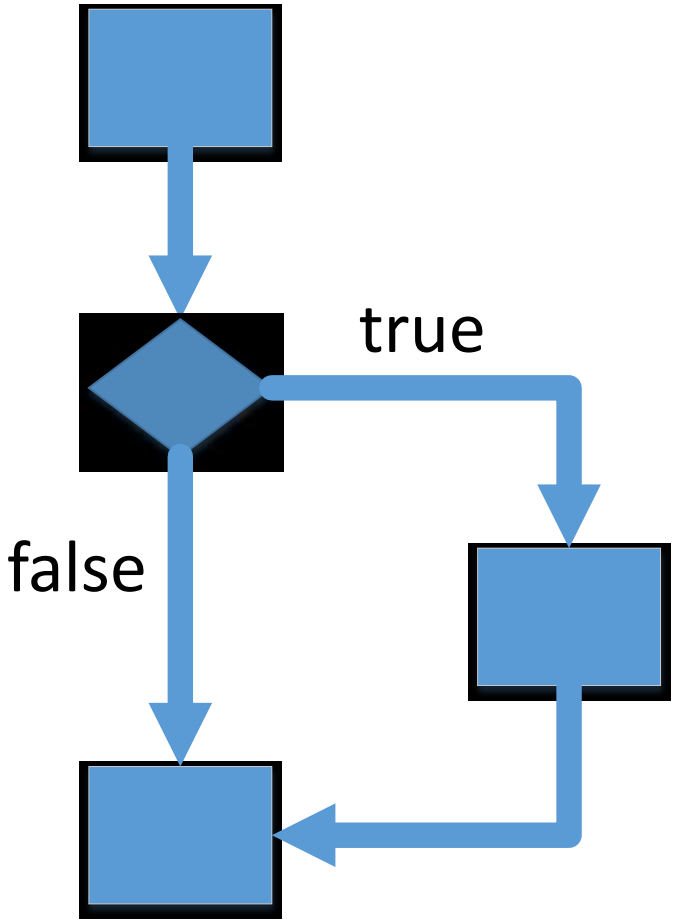
- Also called “iteration” and “looping”
- Allows a block of code to be executed repeatedly until a condition is satisfied



Selection

If

However, it is good practice to go ahead and use them.
You might need to add more statements later and forget to add curly braces.

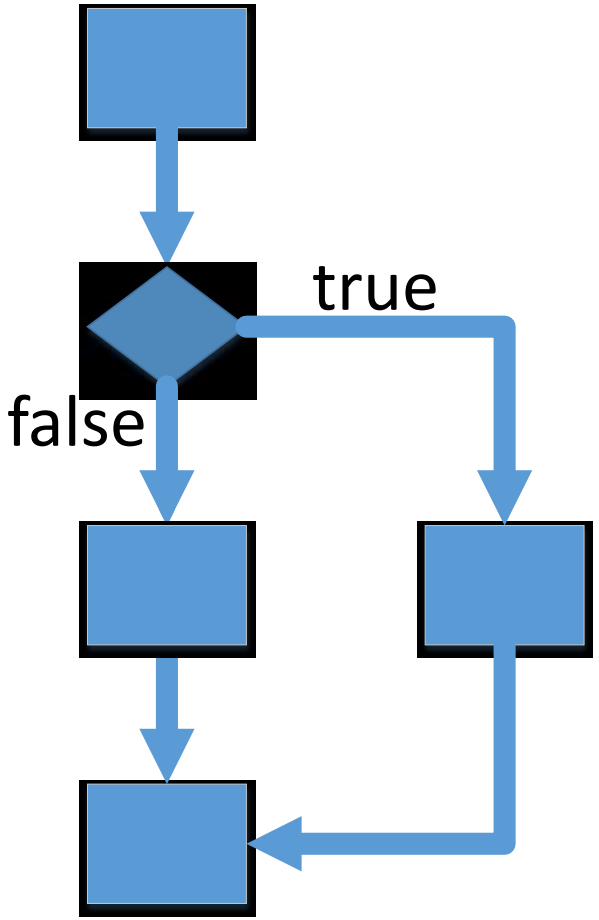



```
if (boolean expression) {

}
```

Note: Curly braces are optional if there is a single statement.

```
if (boolean expression)
   // single statement
```

If-else

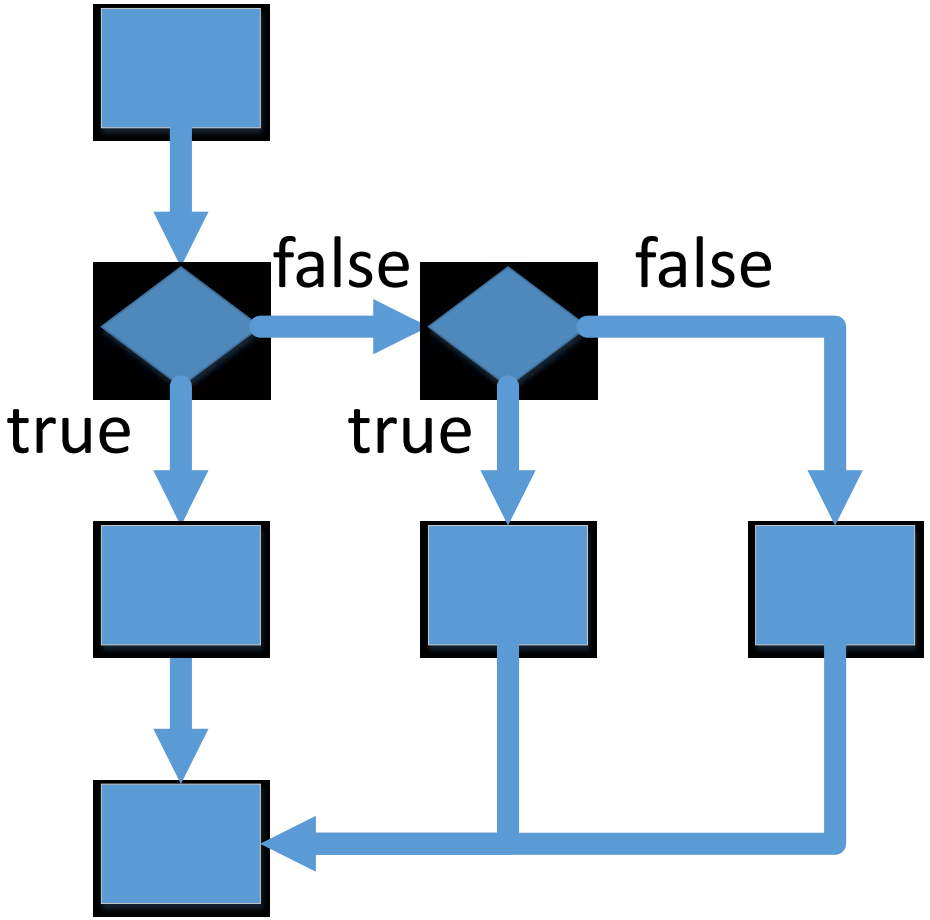



```
if (boolean expression) {
  // do if true
} else {
  // do if false
}
```

Note: Curly braces are optional if there is a single statement.

Multiple if-else

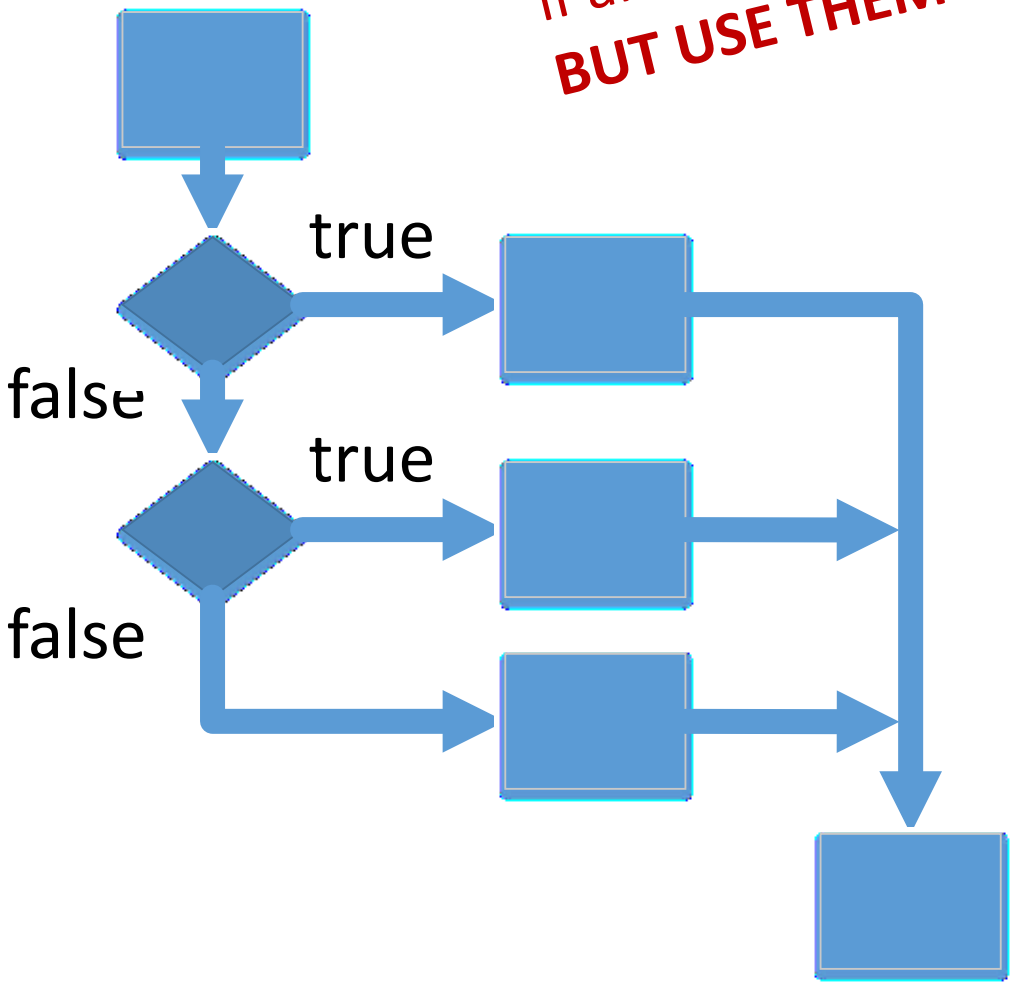
Note: Curly braces are optional if there is a single statement.




```
if (boolean expression) {
  // do stuff
} else if (boolean expr) {
  // do other stuff
} else {
  // do yet other stuff
}
```

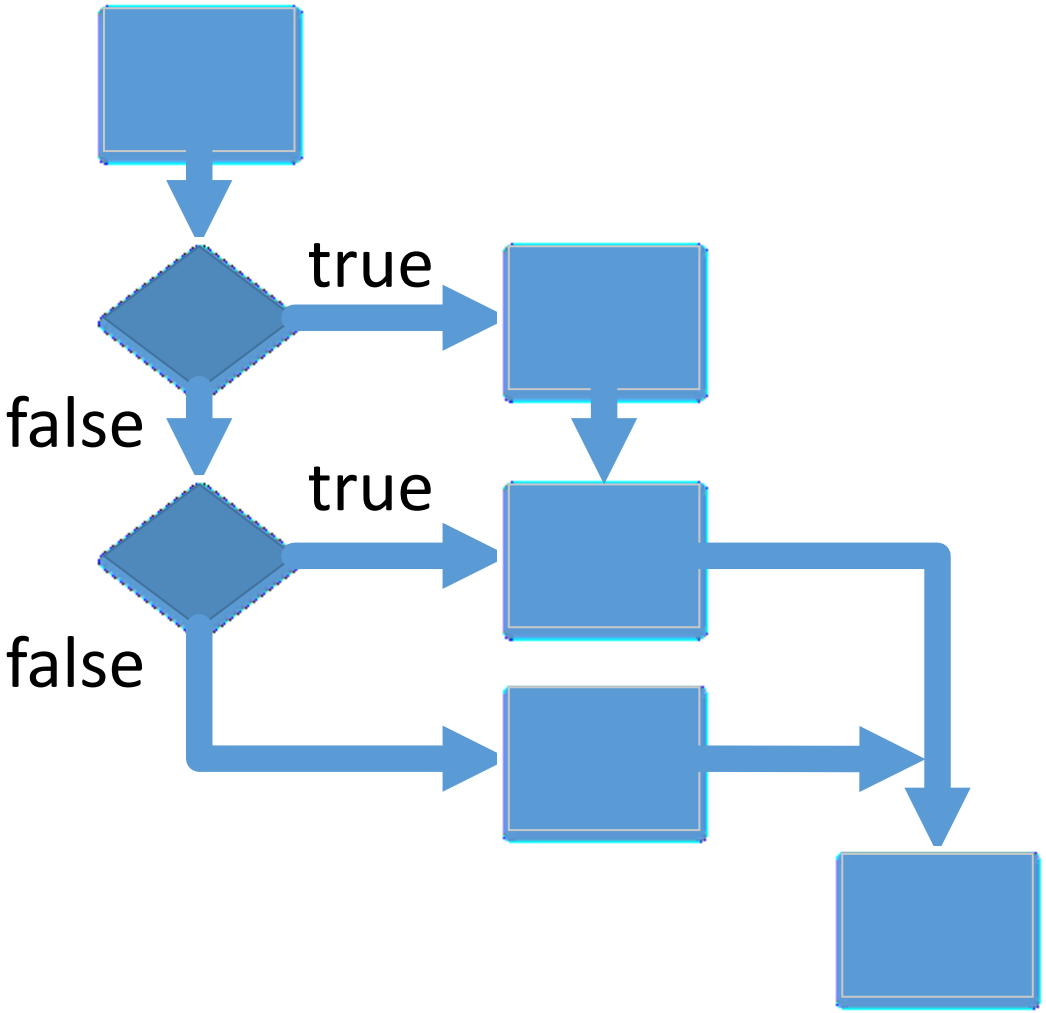
Switch

Note: Curly braces are optional
if all cases are single statements.
BUT USE THEM ALWAYS.




```
switch (value) {
  case val1:
        // do stuff
        break;
  case val2:
        // do other stuff
        break;
  default:
        // yet other stuff
}
```

Switch with fall-through



Note: Curly braces are optional if all cases are single statements. ***BUT USE THEM ALWAYS.***

```
switch (value) {
  case val1:
        // do stuff
  case val2:
        // do other stuff
        break;
  default:
        // yet other stuff
}
```

Switch

- Value used for comparison must be an integer, char, or enumeration.
  - enumeration := identifier that maps to an integer
- When a matching case is found, code is executed until encountering a break. This can result in code for multiple cases executing *(fall-through).*
  - Tip: when debugging, check for missing or mistaken break statements.

Repetition

Parts of a Loop

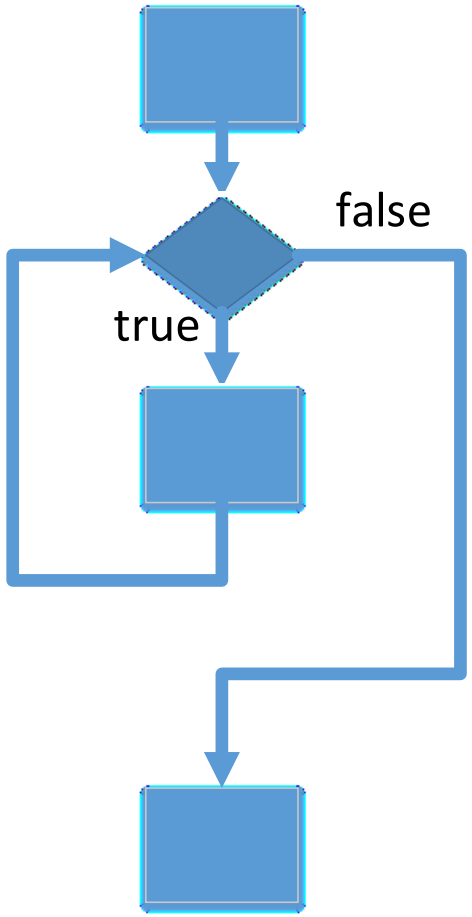
- Initialization
  - Usually of a loop control variable
- Continuation condition
  - Boolean expression specifying when to continue executing the loop
- Update
  - Usually of a loop control variable
- Loop Body
  - Sequence of statements to execute

Iteration := one execution of a sequence of operations or instructions in a repetition.
“The program crashes on the 5th iteration”, “How many iterations has it done, so far?”

Creating a Loop

1. What do things have to look like before entering the loop?
   - Initialization (control variable)
   - Initialization (loop body)
2. How do you exit/end the loop?
   - Control variable
   - Continuation condition
   - Interrupt: break, return, exception
3. What do you do to ensure you eventually exit/end the loop?
   - Update
4. What does the loop do?
   - Loop Body

While

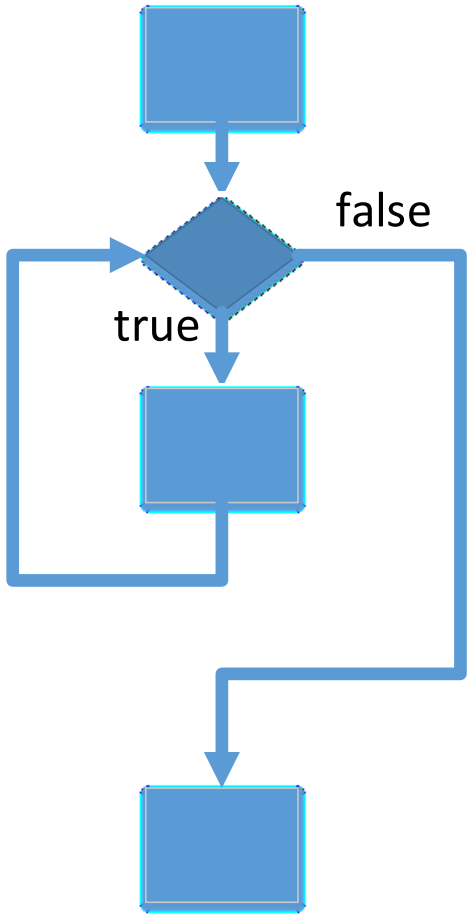



```
// initialize
while (continuation condition) {
        // loop body
        // update
}
```

Note: Curly braces are optional if there is a single statement.

For

Note: Curly braces are optional if there is a single statement.



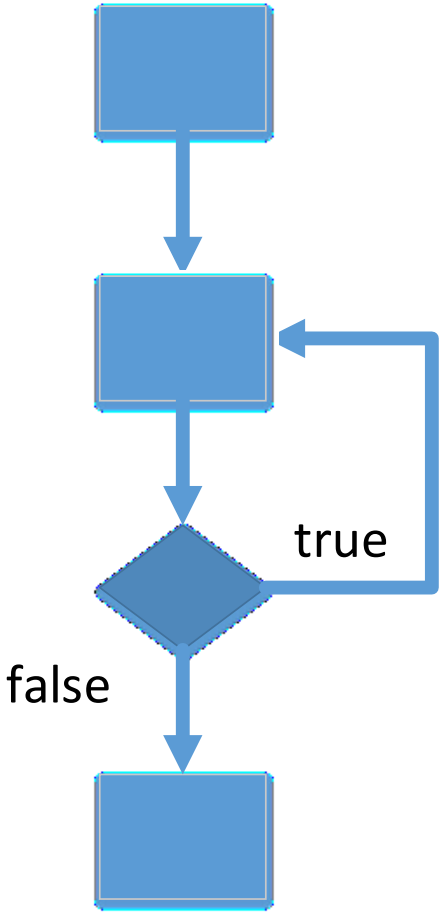

```
// A. initialize
// B. continuation condition
// C. update

for(A; B; C) {
        // loop body
}
```

Bad style to update control variable inside loop.

Types of control

- Counting
  - Control variable increments by set amount each iteration.
- Sentinel
  - Control variable is set to a value obtained during the loop each iteration.
- Flag
  - Control variable is a Boolean variable that represents a condition each iteration.
  - Special case of sentinel

Do While




```
// initialize
do {
        // loop body
        // update (can be initialization)
} while (continuation condition);
```

Note: Curly braces are optional if there is a single statement.

Loops

Loop type	Minimum times through loop	Maximum times through loop
While	0	[redacted]
For	0	[redacted]
Do While	1	[redacted]

Loop type	Initialization	Update
While	Outside of loop	Within loop body
For	Built into statement	Built into statement
Do While	Outside of loop	Within loop body

Loop conversion

- Most loops can be converted to any other type of loop.
  - What can’t? (Well you could but it would be bad practice)
- Good exercise that can help you build good loop structures.
- Remember the guidance for building a loop:
  - Initialize control variable(s)
  - Specify continuation condition
  - Specify loop body
  - Define update for control variable

The Increment and Decrement Operators

- ++ is the increment operator.

  It adds one to a variable.

  `val++;` is the same as `val = val + 1;`

- ++ can be used before (prefix) or after (postfix) a variable:

  ```
  ++val;      val++;
  ```

The Increment and Decrement Operators

- `--` is the decrement operator.

  It subtracts one from a variable.

  `val--;` is the same as `val = val - 1;`

- `--` can be also used before (prefix) or after (postfix) a variable:

  ```
  --val;      val--;
  ```

Prefix vs. Postfix

- `++` and `--` operators can be used in complex statements and expressions
- In prefix mode (`++val, --val`) the operator increments or decrements, *then* returns the value of the variable
- In postfix mode (`val++, val--`) the operator returns the value of the variable, *then* increments or decrements

Prefix vs. Postfix - Examples

```
int num, val = 12;
cout << val++; // displays 12,
               // val is now 13;
cout << ++val; // sets val to 14,
                // then displays it
num = --val;   // sets val to 13,
               // stores 13 in num
num = val--;   // stores 13 in num,
               // sets val to 12
```

Deciding Which Loop to Use

- The `while` loop is a conditional pretest loop
  - Iterates as long as a certain condition exits
  - Validating input
  - Reading lists of data terminated by a sentinel
- The `do-while` loop is a conditional posttest loop
  - Always iterates at least once
  - Repeating a menu
- The `for` loop is a pretest loop
  - Built-in expressions for initializing, testing, and updating
  - Situations where the exact number of iterations is known

Boolean Expressions

Boolean Values

- Logically
  - True
  - False
- C++
  - Represented by an integer in the background
    - false is 0
    - true is literal 1 by default
      - Any non-zero value is *truthy*

Where it is a problem

- Some functions that are expected to be Boolean actually return an int.
  - int isalnum (int c);
  - Check if character is alphanumeric (a decimal digit or an uppercase or lowercase letter).

- What some students in the past have done

```
char c = 'z';
if (isalnum(c) == true) {
  // do something
}
```

  - Does not always work

Solution

- Don’t compare Boolean-ish functions with true or false.
  - Boolean-ish: return a value which is not strictly a Boolean value (e.g. int), but is used as if it returns a Boolean.

- Just use the value directly as a Boolean without comparing it.

- What you should do

```
char c = ‘z’;
if (isalnum(c)) {
  // do something
}
```

  - Always works

Boolean Operators

- And: &&
- Or: ||
- Not: !

Boolean Expressions

- **p && q** is true if and only if both p and q are truthy (not 0/false)
  - **&&** is Boolean AND
  - **&** is bitwise AND operation (does not produce a Boolean value)
- **p || q** is true if and only if either p or q, or both, is truthy (not 0/false).
  - **||** is Boolean OR
  - **|** is bitwise OR operation (does not produce a Boolean value)
- **!p** is true if and only if p is false
  - **!p** is false if and only if p is truthy (not 0/false)
  - **~** is bitwise negation (1's complement, does not produce a Boolean value)

Boolean Operators-Examples

```
int x = 12, y = 5, z = -4;
```

`(x > y) && (y > z)`	`true`
`(x > y) && (z > y)`	`false`
`(x <= z) || (y == z)`	`false`
`(x <= z) || (y != z)`	`true`
`!(x >= z)`	`false`

Boolean Operator-Notes

- **!** has highest precedence, followed by **&&**, then **||**
- If the value of an expression can be determined by evaluating just the sub-expression on left side of a logical operator, then the sub-expression on the right side will not be evaluated (*short circuit evaluation*)