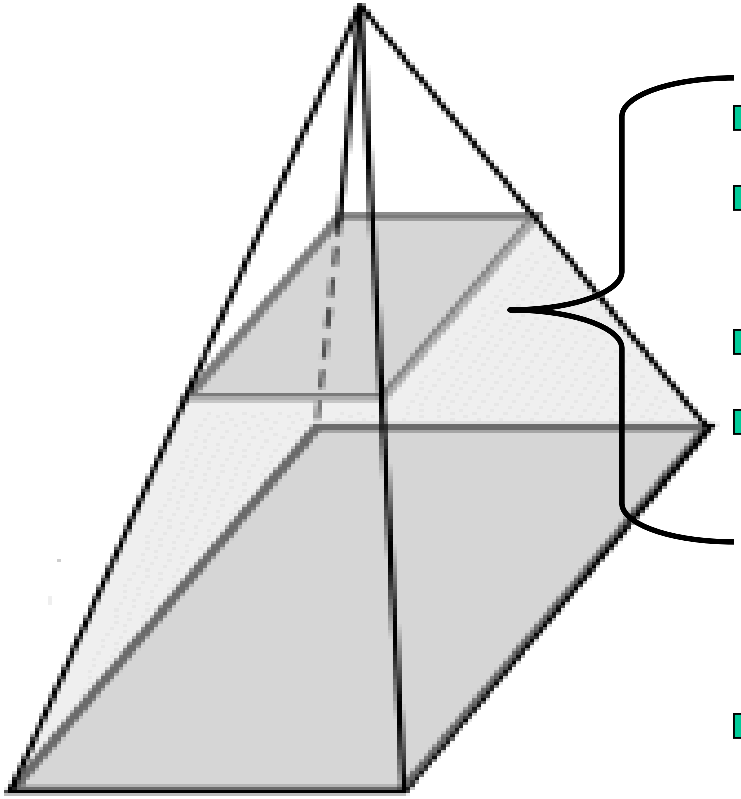
programming - C++

Spring 2002

Thursday, March 21

Keeping Perspective



Basics Covered…….

- Structure of a C++ Program
- Variables, Data Types and Constants
- Operators
- Primitive I/O Operations Console Communication

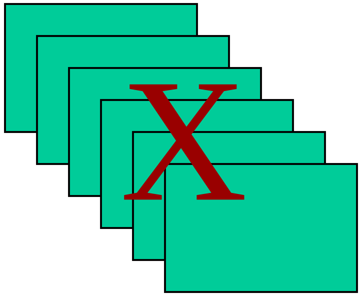
Today……..

- Control Structures

Control Structures

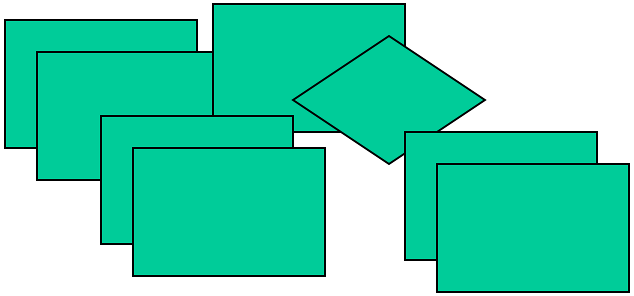
C++ provides **control structures** to manage the logic flow of a program.

A program is rarely limited to a sequence of linear instructions.



During execution, code may need to:

- Repeat
- take decision paths
- branch



Control Structures

Selection

Take action based on the value of one or more constants/variables
if, if else, else if,switch-case

Repetition

Repeat a programming instruction while a condition remains true
while, do while, for

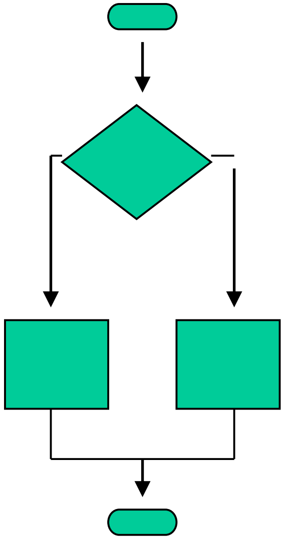
Bifurcation (Branching)

Leave a loop even if the condition for its end is not fulfilled
break, continue, goto, exit

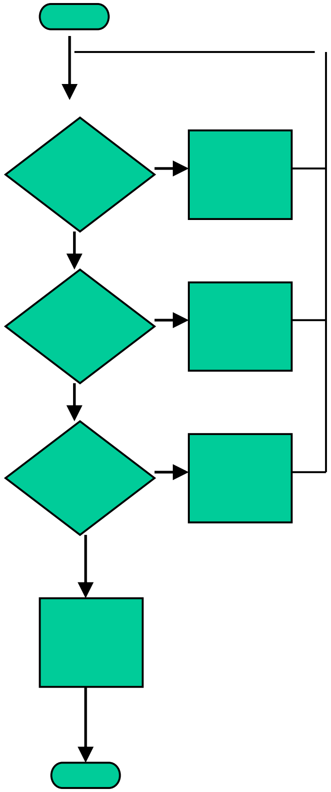
programming - C++
Spring 2002

Control Structures illustrated

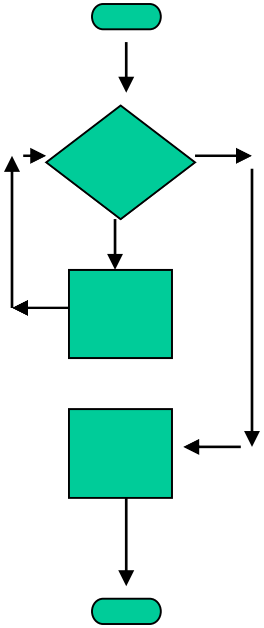
Selection



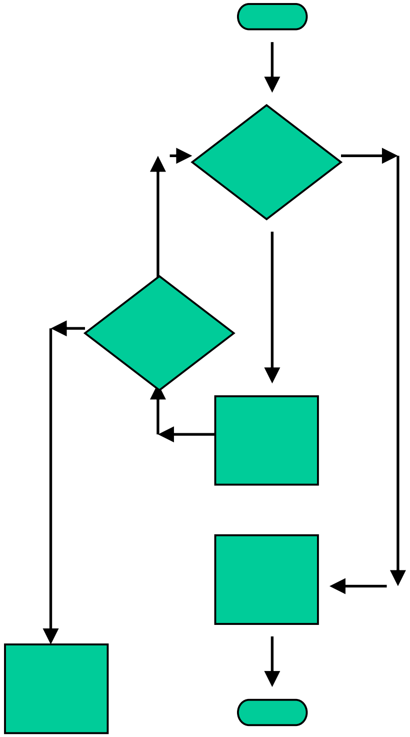
Selection



Repetition



Branching



programming - C++

Spring 2002

Control Structures
if, else, else if

Terms and Concepts

- Condition ( )
- Logical Expressions T/F
- Compound Expressions
- Multiple Statements
  - {} Curly Braces
- Logical Operators
- Relational Operators

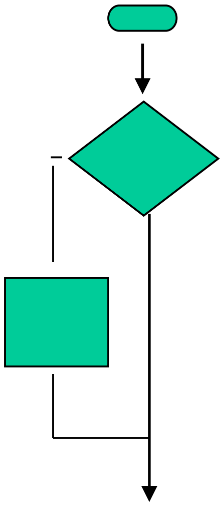
```
width_ok = width < 15;
too_high = height > 13;
load_ok = weight < 45 && weight/(width * Length) < 0.2;
```

```
if (width_ok && weight !too_high && load_ok)
    cout <<  “OK to cross bridge\n”;
else
{
cout << “Do not cross bridge!\n”;
   if (!load_ok)
       cout << “Excessive weight or load factor.\n”;
   if (!width_ok || too_high);
       cout << “Too wide or too high\n”;
   if (height = = 13)
       cout << “Height is at the borderline.\n”;
}
```

Control Structures

syntax - if

One Way Selection



```
if (condition)
      statement
```

condition

logical expression

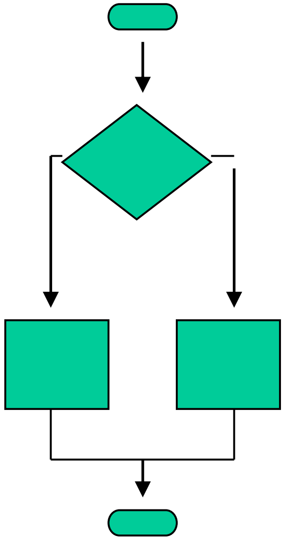
```
if (width < 30)
    cout <<  "OK to cross bridge.\n";
```

statement

Control Structures

*syntax – **if else***

Two-Way Selection



```
if  (condition)
      statement1
else
      statement2
```

condition

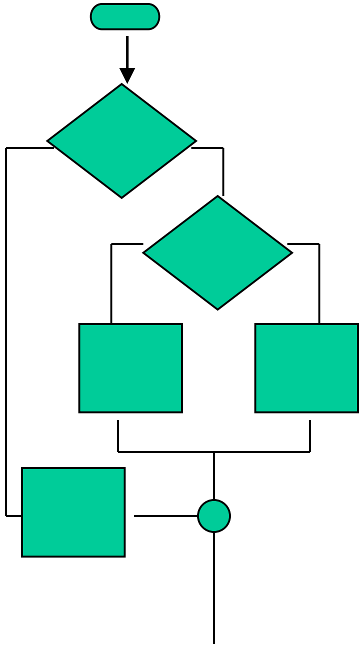
logical expression

```
if (width < 30)
    cout <<  “OK to cross bridge.\n”;
else
    cout <<  “NOT ok to cross bridge.\n”;
```

statements

Control Structures
syntax – else if

Multi-Way
Linear
Selection
AKA Nested if



```
if  (condition)
   statement1
else
      if  (condition2)
            statement2
.
.
else
      statement n
```

```
if (width_ok && !too_high )
     cout <<  “OK to cross bridge.\n”;
else
    if (!load_ok)
         cout << “Excessive weight.\n”;
    else
```

Control Structures using a compound expression

```
Operand1 operator Operand2
```

condition

compound expression

```
if (width_ok && weight !too_high && load_ok)
    cout <<  “OK to cross bridge.\n”;
else
    cout <<  “NOT ok to cross bridge.\n”;
```

Control Structures using multiple statements

```
{
 statement1
 statement2
}
```

```
if (width_ok && weight  !too_high && load_ok)
     cout <<  “OK to cross bridge.\n”;
else
    {
    cout <<  “NOT ok to cross bridge.\n”;
    cout <<  “GO HOME!\n”;
    }
```

Multiple statements

Control Structures review

e 1.

```
(width > 30)
```

d 2.

```
width_ok && !too_high
```

b 3.

```
cout <<  “OK to cross bridge.\n”;
```

a 4.

```
{
   cout <<  “NOT ok to cross bridge.\n”;
   cout <<  “Go Home!\n”;
}
```

c 5.

```
if (width_ok && !too_high )
    cout <<  “OK to cross bridge.\n”;
else
   if (!load_ok)
       cout << “Excessive weight.\n”;
```

a. Multiple statement

b. Statement

c. Multi-way selection

d. Compound expression

e. Condition

The NOT operator accepts one input; if that input is TRUE, it returns FALSE, and if that input is FALSE, it returns TRUE. In C and C++ NOT is written as !. NOT is evaluated prior to both AND and OR.

Control Structures using relational operators

Relational operators are used in expressions to enable Comparisons. The condition returns a True or False.

Equal	= =	**(5 = = 4)**	would return **false**
Not equal	!=	**(5 != 4)**	would return **true**
Greater than	>	**(5 > 4)**	would return **true**
Less than	>=	**(5 < 4)**	would return **false**
Greater or equal than	>=	**(5 >= 4)**	would return **true**
Less or equal than	<=	**(5 <= 4)**	would return **false**

Note: In many programming languages, " = " is used to assign a value to a variable and as a Relational Operator to test a condition.

Control Structures using logical operators

Logical operators are used to evaluate compound expressions and obtain a single result.

&&	(AND)	- Both
||	(OR)	- Either
!	(NOT)	- Reverse

```
Operand1 operator Operand2
```

```
if ((type = = ‘a’ || age > 25) && years !< 5)

   cout << “Qualifies for Discount";
```

Control Structures compound expression evaluation

The table below shows the evaluation returned by logical operators for possible operand values.

<table>
<tr><th>First Operand</th><th>Second Operand</th><th colspan="2">Result</th><th colspan="2">Result</th></tr>
<tr><th>a</th><th>b</th><th>(a && b)</th><th>!(a && b)</th><th>(a || b)</th><th>!(a || b)</th></tr>
<tr><td>true</td><td>true</td><td>true</td><td>false</td><td>true</td><td>false</td></tr>
<tr><td>true</td><td>false</td><td>false</td><td>true</td><td>true</td><td>false</td></tr>
<tr><td>false</td><td>true</td><td>false</td><td>true</td><td>true</td><td>false</td></tr>
<tr><td>false</td><td>false</td><td>false</td><td>true</td><td>false</td><td>true</td></tr>
</table>

Control Structures maintaining good practices

When coding control statements, always be sure you…..

- Understand the problem
- Use top-down design – decompose the problem
- Psuedo-code
- Code
- Test & Debug

Control Structures language comparison

	C++	JavaScript	Java	C#
Logical Operators	√	√	√	√
Relational Operators	√	√	√	√
Compound Statements	√	√	√	√
Compound Expressions	√	√	√	√
Control Structure Syntax				
if, if else, else if	√	√	√	√
while	√	√	√	√
do while	√	√	√	√
for	√	√	√	√
for in		√		
for each				√
switch case	√	√	√	√
break	√	√	√	√
continue	√	√	√	√
goto	√			√

Control Structures review

Mark the boxes below as True or False indicating the return value of the expression. Last 2 are fill in the blanks.

Assume **a=2**, **b=3** and **c=6**

[T] **1. (a*b >= c)**

[F] **2. (b+4 > a*c)**

[T] **3. ((b=2) = = a)**

4. Name two other types of operators discussed in previous classes. __arithmetic_ _assignment_

5. && is an example of a _logical____ **operator while = = is a** _relational_ **operator.**

In many compilers previous to the publication of the ANSI-C++ standard, as well as in the C language, the relational operations did not return a **boolean** value of **true** or **false**, rather they returned an **int** as The result with a value of **0** represent **"false"** and a value different from **0** (generally **1**) to represent **"true"**.

Control Structures debug program

```
//wages.cpp
#include <iostream.h>
#include "\ourtools.h"
void main()
{
const float MIN_WAGE = 5.35;
int hours;
float rate, wages;

cout << "Enter hours worked and hourly rate: “;
cin >> hours >> rate;
{
if (hours >= 0 & rate >= MIN-WAGE)
    // valid inputs for hours and rate
    if (hour <= 40)
        wages == hours * rate;
    else
        wages = 40*rate + (hours-40)*2.0*rate;
    fixed-out (cout, 2);
    cout << "Wages <<  == $” << wages << endl;
    }
else // hours and/or rate invalid
   cout << "INPUT ERROR(S).\n";
}
```