H.O.#3
Fall 2015
Gary Chan

# Program Flow Control

# Topics

- `if-else` and `if-else if` statements
- Operator precedence
  - bitwise operators
- `switch` statements
- `while` statements
- `do-while` statements

# Three Program Structures

- Sequence - executable statements which the computer processes in the given order
- Choice - sequence(s) selected depending on some condition

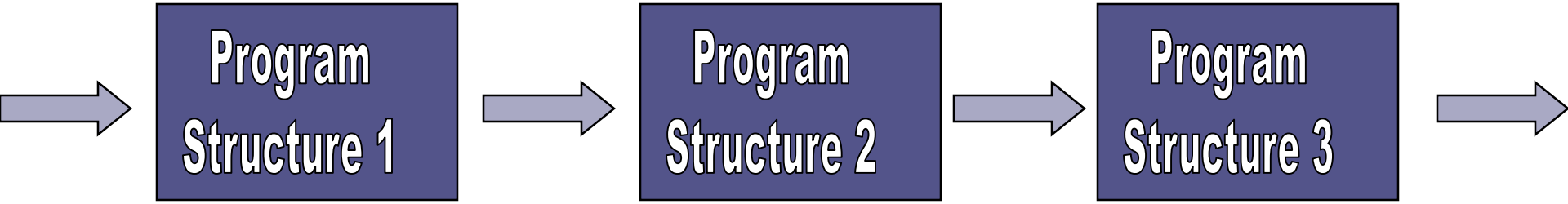
```
if <condition exists>{
  <do P>
}
```

- Iteration - repetitively executed sequences

```
while <condition exists>{
  <do P>
}
```

# Sequence

- **It is natural to write a program as a sequence of program structures such as sequences, choices, and iterations**



# Choice Constructs

- Provide
  - Ability to control whether a statement list is executed
- Two constructs
  - `if` statement
    - `if`
    - `if-else`
    - `if-else if`
  - `switch` statement

# The Basic `if` Statement

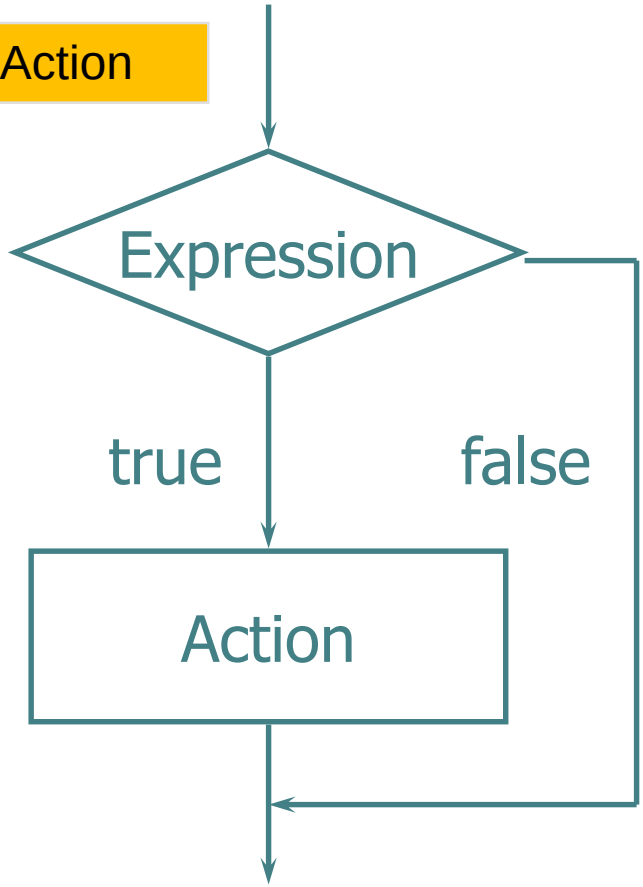
- Syntax

Result is non-zero, do Action

```
if(Expression)
    Action
```

- Example: absolute value

```
if(value < 0)
    value = -value;
```



# Absolute Value

```
// program to read number & print its absolute value
#include <iostream>
using namespace std;
int main(){
  int value;
  cout << "Enter integer: ";
  cin >> value;
  if(value < 0)
      value = -value;
  cout << "The absolute value is " << value << endl;
  return 0;
}
```

# Choice (`if`)

* Put multiple action statements within braces

```
if <it's raining>{
    <take umbrella>
    <wear raincoat>
}
```

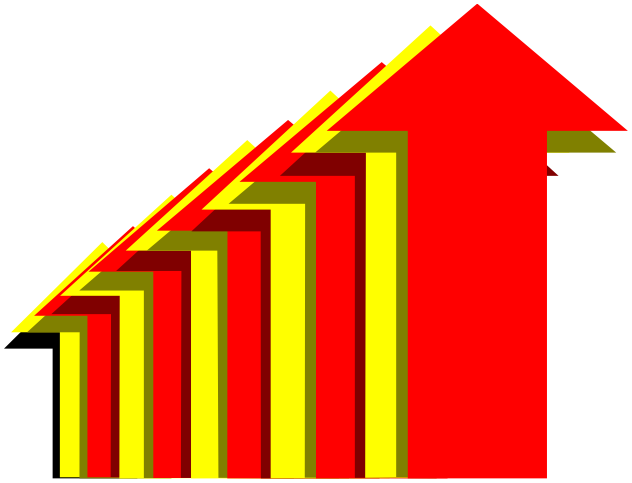
```
if(-1)
    cout << -1 << endl;
if( 2 )
    cout << 2 << endl;
if( 0 )
    cout << 0 << endl;

if( ! 4 )    // or !-4
    cout << 4 << endl;
```

executed

Unexecuted

# Sorting Two Numbers

```
int value1;
int value2;
int temp;
cout << "Enter two integers: ";
cin >> value1 >> value2;
if(value1 > value2){
  temp = value1;
  value1 = value2;
  value2 = temp;
}
cout << "The input in sorted order: "
     << value1 << " " << value2 << endl;
```



# Relational Operators

Relational operators are used to compare two values

| Math | C++ | Plain English |
|---|---|---|
| $=$ | `==` | equals [example: `if(a==b)` ] |
| | | [ `(a=b)` means put the value of `b` into `a` ] |
| $<$ | `<` | less than |
| $\leq$ | `<=` | less than or equal to |
| $>$ | `>` | greater than |
| $\geq$ | `>=` | greater than or equal to |
| $\neq$ | `!=` | not equal to |

# Relational Expressions

Examples:

```
numberOfStudents < 200

20 * j == 10 + i
```

# Operator Precedence

**Which comes first?**

Answer: * / %

+ -

< <= >= >

== !=

=

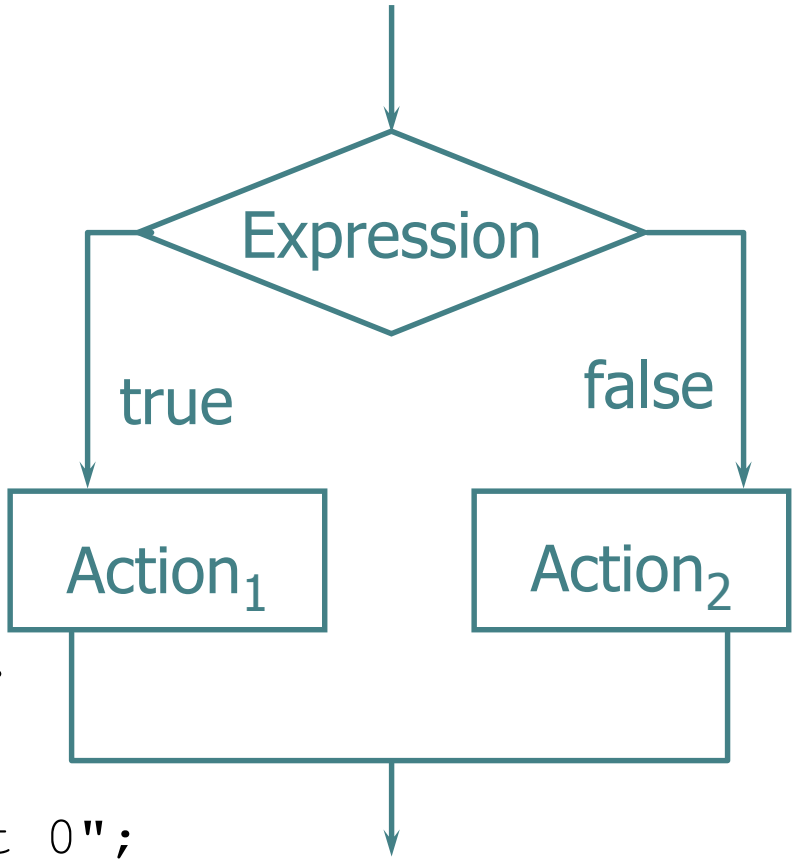
# The `if-else` Statement

- Syntax of if-else

```
if <it's sunny>{
  <walk cat>
}
else{
  <take cat with umbrella>
}
```

- Example

```
if(v == 0)
        cout << "v is 0";
else
        cout << "v is not 0";
```



# Finding the Bigger Value

```
int value1;
int value2;
int larger;
cout << "Enter two integers: ";
cin >> value1 >> value2;
if(value1 > value2)
  larger = value1;
else
  larger = value2;
cout << "Larger of inputs is: " << larger << endl;
```

# Selection

- Often we want to perform a particular action depending on the value of an expression
- Two ways to do this
  - `if-else-if` statement
    - `if-else` statements "glued" together
  - `switch` statement
    - An advanced construct

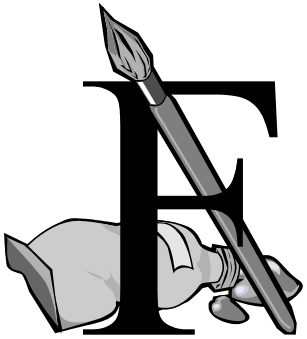
# if-else if Statements

```
if <Mon, Wed, or Fri AM>{
    <goto MATH>
}
else if <Tues, Thurs PM>{
    <goto COMP2012H>
}
else if <1PM or 7PM>{
    <eat>
}
else{
    <sleep>
}
```

# `if-else-if` Statement

- Example

```
if(score >= 90)
     cout << "Grade = A" << endl;
else if(score >= 80)
     cout << "Grade = B" << endl;
else if(score >= 70)
     cout << "Grade = C" << endl;
else if(score >= 60)
     cout << "Grade = D" << endl;
else
     cout << "Grade = F" << endl;
```

# switch Statement

```
switch(int(score)/10){
    case 10:    // Note: empty here
    case 9:     cout << "Grade = A" << endl;
                break;
    case 8:     cout << "Grade = B" << endl;
                break;
    case 7:     cout << "Grade = C" << endl;
                break;
    case 6:     cout << "Grade = D" << endl;
                break;
    default:    cout << "Grade = F" << endl;
}
```

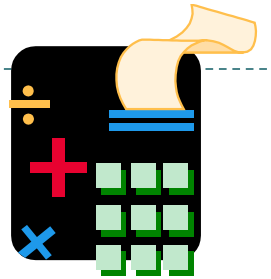
# Remember to `break` if you do not want `case` to falls through

```
int i=4;

switch(i){
 case 1:
   cout << 1 << endl;
 case 2:
   cout << 2 <<endl;
 case 4:
   cout << 4 << endl;
 case 5:
   cout << 5 << endl;
 default:
   cout << "default" << endl;
 }
```

Output:

```
4
5
default
```

# A Calculator Program



```
int left;
int right;
char oper;
cout << "Enter simple expression: ";
cin >> left >> oper >> right;
cout << left << " " << oper << " " << right
 << " = ";
switch (oper) {
  case '+' : cout << left + right << endl; break;
  case '-' : cout << left - right << endl; break;
  case '*' : cout << left * right << endl; break;
  case '/' : cout << left / right << endl; break;
  default: cout << "Illegal operation" << endl;
}
```

# `enum` type

- Assigning a list of items to some numeric values
  - Define a set of alternative values for some attributes
- Usually for code readability
- By default, the first enumerator is assigned the value 0
- Enumerators are `const` values/objects
- An object of enumeration type may be initialized or assigned only by one of its enumerators or by another object of the same enumeration type
- An enumerator value need not be unique (e.g.,

```
enum Forms {shape = 1, sphere, cylinder = 2,
  polygon};
```

```
#include <iostream>
using namespace std;


int main(){

  enum fruit {APPLE = 2, PINEAPPLE, ORANGE, BANANA};
  fruit snack;

  cout << "What fruit to eat for snack? (2 for apple, 3 for pineapple, 4 for orange,
and 5 for banana)" << endl;
  cin >> (int) snack;    // cin gets int

  switch( snack ){
  case APPLE:
    cout << "Have apple!" << endl;
    break;
  case PINEAPPLE:
    cout << "Have pineapple!" << endl;
    break;
  case ORANGE:
    cout << "Have orange!" << endl;
    break;
  case BANANA:
    cout << "Have banana!" << endl;
    break;
  default:
    cout << "No such choice." << endl;
    break;
  }

  exit( 0 );
}
```

```
cssu5:> a.out
What fruit to eat for snack? (2 for
apple, 3 for pineapple, 4 for orange, and
5 for banana)
2
Have apple!
cssu5:> a.out
What fruit to eat for snack? (2 for
apple, 3 for pineapple, 4 for orange, and
5 for banana)
6
No such choice.
```

# A Boolean Type

- C++ contains a type named `bool` which can have one of two values
  - `true` (corresponds to non-zero value 1)
  - `false` (corresponds to zero value 0)
- Boolean operators can be used to form more complex conditional expressions
  - The and operator is `&&`
  - The or operator is `||`
  - The not operator is `!`
  - A conditional expression terminates early once the result is known
- Warning
  - `&` and `|` are also operators (bitwise operator)
  - << and >> are also bitwise operators
  - Discussed later

# A Boolean Type

- 4 bytes, takes on values 1 or 0
  - `!true = 0; !false = 1`
- Example logical expressions

```
bool P = true;
bool Q = false;
bool R = true;
bool S = P && Q;
bool T = !Q || R;
bool U = !(R && !Q);
bool V = ( x > y );
```

# More Operator Precedence

- Precedence of operators (from highest to lowest)
  - Parentheses ( ... )
  - Unary operators !
  - Multiplicative operators * / %
  - Additive operators + -
  - Relational ordering < <= >= >
  - Relational equality == !=
  - Logical and &&
  - Logical or ||
  - Assignment =

# More Operator Precedence

- Examples

```
5 != 6 || 7 <= 3
```

```
(5 !=6) || (7 <= 3)
```

```
5 * 15 + 4 == 13 && 12 < 19 || !false == 5 < 24
```

false true true

# Bitwise Operators

- Numbers are represented as bits in computer
  - E.g., 2 = 0010, 8 = 01000, 12 = 01100
- A programmer may manipulate the bits using operators
  - >> (right-shift)
  - << (left-shift)
  - & (and)
  - | (or)
  - ~ (not)

```
#include <iostream>
using namespace std;

int main(){

  int n = 15;     /* n = 00001111 */
  int m = 18;     /* m = 00010010 */

  cout << (n << 2) << endl;      // get 00111100
  cout << (n >> 2) << endl;      // get 00000011
  cout << ( n | m ) << endl;     // get 00011111
  cout << ( n & m ) << endl;     // get 00000010
  cout << ( ~m & 7 ) << endl;    // get 00000101

  int mask = 0;
  for(int i = 0; i <= 7; i++ )
    mask = mask | (1 << i);
  cout << mask << endl;   // mask is 255: the last 8 bits are 1
}
```

Output answers: 60, 3, 31, 2, 5

# The `if` code region is entered only if the expression is non-zero

```
int i=3;
 if( !i )
   cout << i << endl;
 if( ~i )
   cout << i << endl;

 if( 13 && -12 )     // similar to i=13; j=-12, if (i && j)
   cout << "entered" << "\n";
```

Unexecuted

Executed

Executed

# Nested `if` Statements

- Nested means that one complete statement is inside another
- Example:

```
if <it is Tuesday>{
  if <it is time for class>{
   <go to COMP2012H>
  }
  <call your friends>
}
```

# "Dangling Else" Problem

- Problem: Nested `if` statements can seem ambiguous in their meaning.
- What is the value of `c` after the following is executed?

```
int a=-1, b=1, c=1;
if(a>0)
   if(b>0)
      c = 2;
else
   c = 3;
```

# "Dangling Else" Problem

- C++ groups a dangling else with the most recent `if`.
- The following indentation shows how C++ would group this example (answer: `c=1`).

```
int a=-1, b=1, c=1;
if(a>0)
   if(b>0)
      c = 2;
   else    // dangling else grouped to nearest if
      c = 3;
```

# "Dangling Else" Problem

- Use extra brackets { } to clarify the intended meaning, even if not necessary.

```
int a=-1, b=1, c=1;
if(a>0){
   if(b>0)
      c = 2;
   else          // parenthesis avoid dangling else
      c = 3;
}
```

```
int a=-1, b=1, c=1;
if(a>0){
  if(b>0)
    c = 2;
}
else
  c = 3;
```

# Shortcut Assignment

- C++ has a set of operators for applying an operation to a variable and then storing the result back into the variable
- Shortcut assignments: `*=, /=, +=, -=, %=`
- Examples

```
int i = 3;
i += 4;                    // i = i + 4
cout << i << endl;         // i is now 7

double a = 3.2;
a *= 2.0;                  // a = a * 2.0
cout << a << endl;         // a is now 6.4

int change = 1265;
change %= 100;             // change = change % 100
cout << change << endl;    // change is now 65
```

# Increment and Decrement

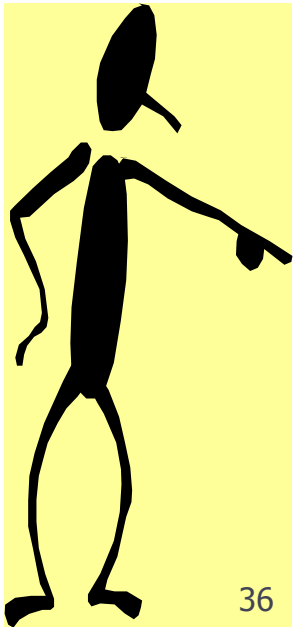
- C++ has special operators for incrementing or decrementing an object by one
- Examples

```
int k = 4;
++k;                        // k=k+1 : k is 5
k++;                        // k=k+1 : k is 6
cout << k << endl;
int i = k++;                // i is 6, k is 7
cout << i << " " << k << endl;
int j = ++k;                // j is 8, k is 8
cout << j << " " << k << endl;
```

# Increment and Decrement

- What is the difference between k++ and ++k?
  - ++k increments first, and the incremented value is used in the expression
  - k++ uses the initial value of k in the expression, and increments afterwards
- Examples

```
int a, b, c, d, k;
k = 3;
a = ++k;        // k=4, a=4
b = --a;        // a=3, b=3
c = b++;        // c=3, b=4
d = c--;        // d=3, c=2
```



# Loops

# Iterative Constructs

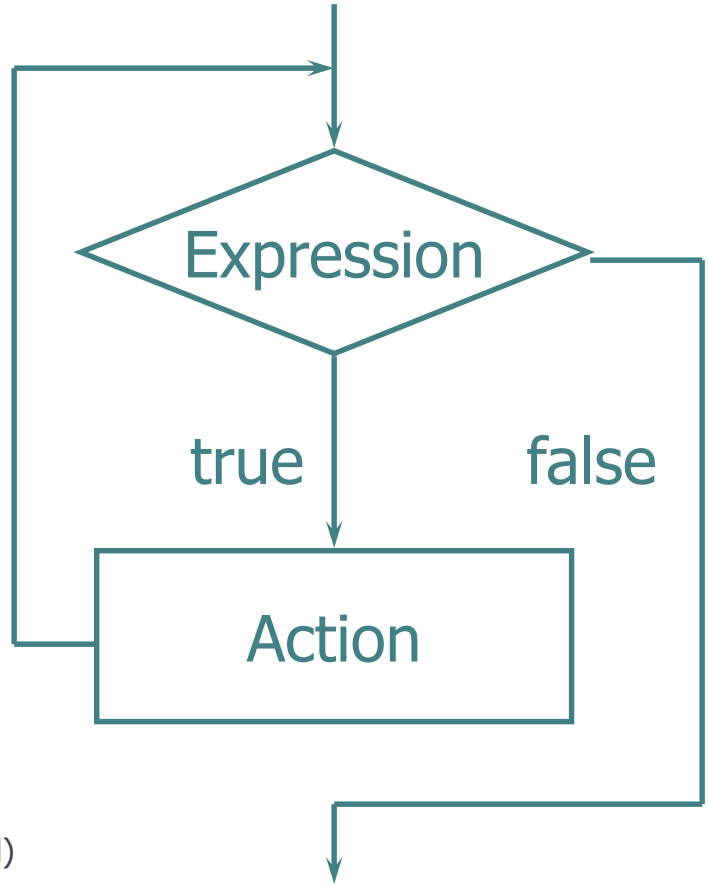
- Provide
  - Ability to control how many times a statement list is executed
- Three constructs
  - `while` statement
  - `for` statement
  - `do-while` statement

# The `while` Statement

- Syntax

```
while (Expression)
    Action
```

- How it works:
  - If Expression is true then execute Action
  - Repeat this process until Expression evaluates to false
- Action is either a single statement or a group of statements within braces



# N! (while)

```
int number, factorial, n;
cout << "Enter number: ";
cin >> number;
factorial = 1;
n = 1;
while(n <= number){
     factorial *= n;
     n++;
}
cout << "The factorial of " << number
     << " is " << factorial << endl;
```

# $2^N$ (while)

```
int number, result, n;
cout << "Enter number: ";
cin >> number;
result = 1;
n = 1;
while(n <= number){
      result *= 2;
      n++;
}
cout << "Two raised to the " << number
      << " power is " << result << endl;
```

1 << number

# Finding the Maximum Input (using `while`)

```
int value=0;    //input value
int max=0;      //maximum value
while(value!=-1){
      cout << "Enter a value (-1 to stop): ";
      cin >> value;
      if(value > max)
            max = value;
}
cout << "The maximum value found"
      << " is " << max << endl;
```

# Averaging numbers

```
#include <iostream>
using namespace std;
int main() {
      int listSize = 0;
      int value;
      double sum = 0.;
      double average;

      cout << "Provide a list of numbers (CTRL-D to stop) " << endl;
      while (cin >> value) {
              sum += value;
              listSize++;
      }
      if(listSize > 0){
              average = sum / listSize;
              cout << "Average: " << average << endl;
      }
      else
              cout << "No list to average" << endl;
      return 0;
}
```

The value of the input operation corresponds to true only if a successful extraction was made

# The `for` Statement

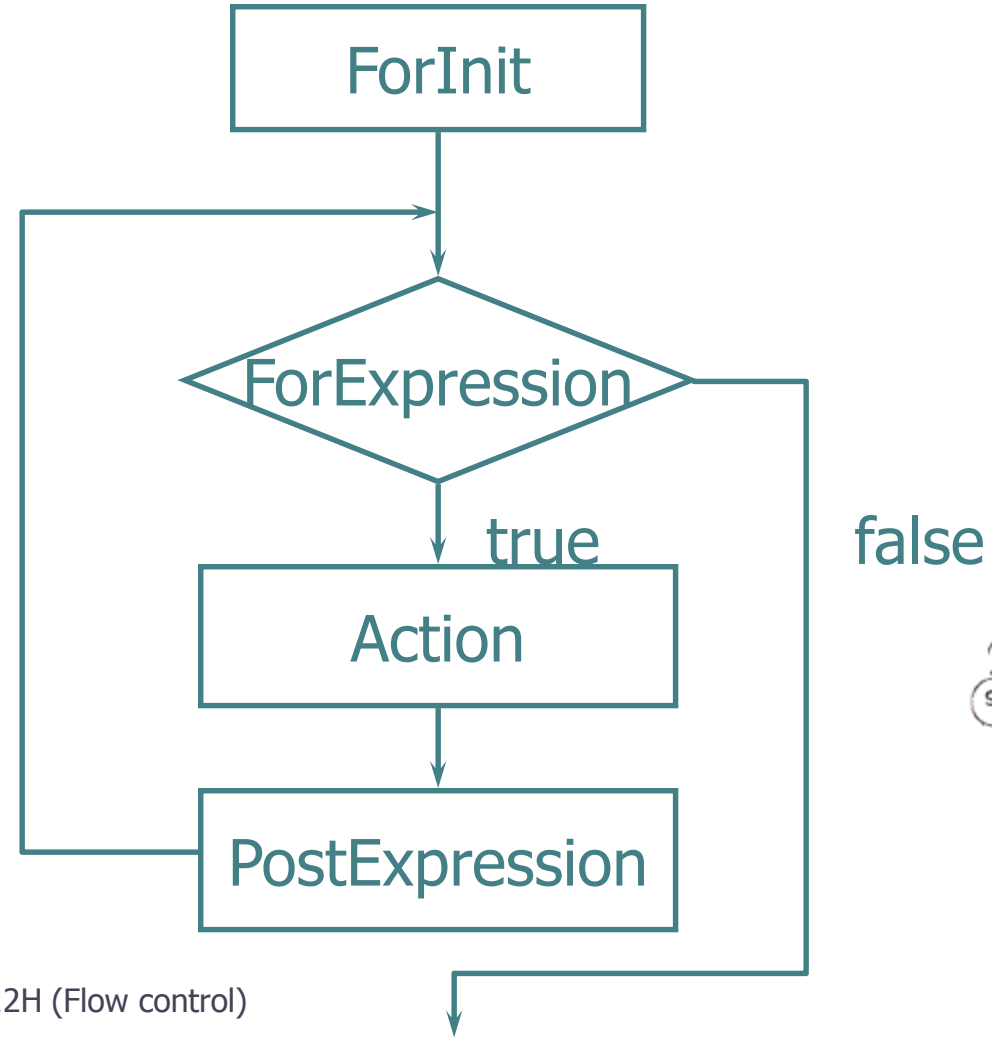
- Syntax

```
for (ForInit; ForExpression; PostExpression)
    Action
```

- How it works:
  - Execute ForInit statement
  - While ForExpression is true
    - Execute Action
    - Execute PostExpression
- Example

```
int i;
for(i=1; i<=20; i++)
    cout << "i is " << i << endl;
```

# Iteration Using `for` Statement




# N! (for)

```
int number, factorial, n;

cout << "Enter number: ";
cin >> number;
factorial = 1;
for(n=1; n<=number; n++)
     factorial *= n;
cout << "The factorial of " << number
     << " is " << factorial << endl;
```

# $2^N$ (for)

```
int number, result, n;

cout << "Enter number: ";
cin >> number;
result = 1;
for(n=1; n<=number; n++)
     result *= 2;
cout << "Two raised to the " << number
     << " power is " << result << endl;
```

# One may put multiple statements in `ForInit` and `PostExpression`

```
for( i = 0, j=0; i < 10; i++, j++ )
   cout <<  i << ", " << j << '\n';
```

```
 for(;;){
   // forever loop
}
```

Output:

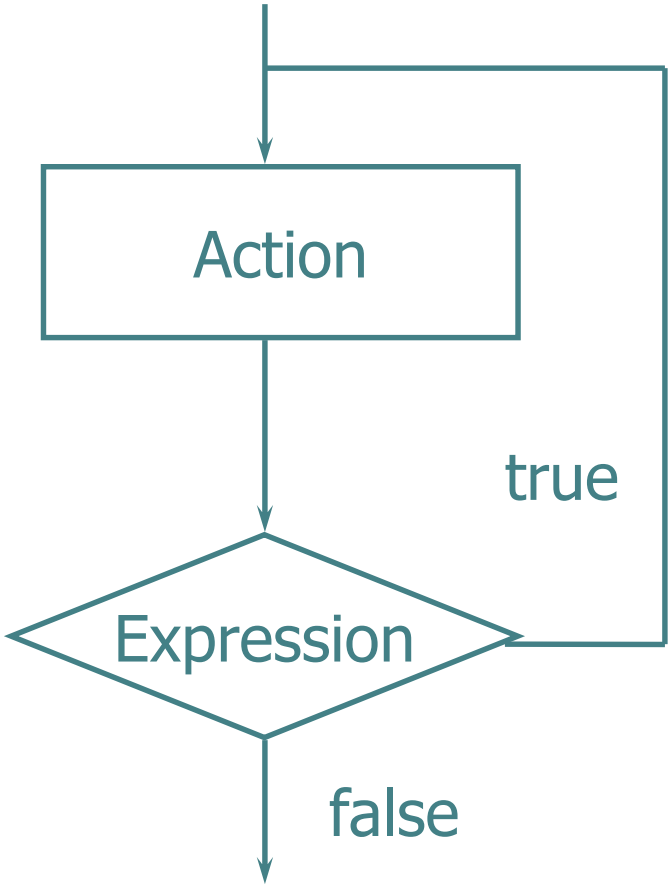
```
0, 0
1, 1
2, 2
3, 3
4, 4
5, 5
6, 6
7, 7
8, 8
9, 9
```

# The Do-While Statement

- Syntax

  `do` Action

  `while` (Expression)
- How it works:
  - Execute Action
  - if Expression is true then execute Action again
  - Repeat this process until Expression evaluates to false
- Action is either a single statement or a group of statements within braces



# N! (do-while)

```
int number, factorial, n;
cout << "Enter number: ";
cin >> number;
factorial = 1;
n = 1;
do{
     factorial *= n;
     n++;
}while(n <= number);
cout << "The factorial of " << number
     << " is " << factorial << endl;
```

# $2^N$ (do-while)

```
int number, result, n;
cout << "Enter number: ";
cin >> number;
result = 1;
n = 1;
do{
     if(number != 0)
          result *= 2;
     n++;
}while (n <= number);
cout << "Two raised to the " << number
     << " power is " << result << endl;
```

# Maximum (do-while)

```
int value;      //input value
int max=0;      //maximum value

do{
     cout << "Enter a value (-1 to stop): ";
     cin >> value;
     if(value > max)
          max = value;
}while(value!=-1);
cout << "The maximum value found is "
     << " is " << max << endl;
```

# Waiting for a Reply

```
char reply;
do{
    //do something
    cout << "Continue(y/n): ";
    cin >> reply;
}while(reply!='n');
```

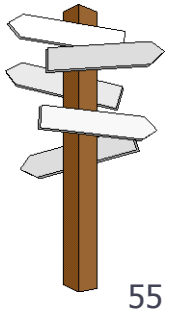
Note that in C++, character is internally represented by an 8-bit number. To compare characters, we use 'n' (note the quotation)

# `break` and `continue`

- A `break` causes the innermost enclosing *loop* (`while, do while, for`) or `switch` to be exited immediately
- A `continue` causes the next iteration of the enclosing `for`, `while`, or `do` loop to begin.
  - In the `while` and `do`, this means that the test part is executed immediately
  - In the `for`, control passes to the increment step.
  - Applied only to loops, not to `switch`

# Which Loop to Use?

- `for` loop
  - Usually best for sums, products, and counting loops.
- `while` loop
  - You want to repeat an action without knowing exactly how many times it will be repeated.
  - You are working with user input
  - There are situations when the action should not be executed.
- `do-while` loop
  - The action should always be executed at least once.
  - Otherwise, the `do-while` loops and `while` loops are used in similar situations.

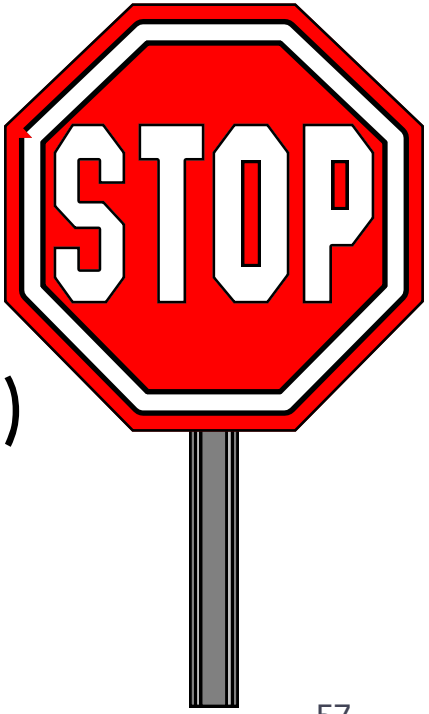


# Iteration

- Key Points
  - Make sure there is a statement that will eventually stop the loop
  - Make sure to initialize loop counters correctly
  - Have a clear purpose for the loop

# How to Stop a Loop

- Known number of iterations before the loop stops (`for`)
- Test for a user-controlled condition before or after each iteration (`while, do-while`)



# Common Loop Errors

```
while(balance >= 0.0);
{
    balance = balance - amount;
}
```

- **This will lead to an infinite loop!**

```
for(n=1; n<=count; n++);
{
    cout << "hello" << endl;
}
```

- **"hello" only printed once!**

# Potential Loop Errors

```
while(balance != 0.0){
    balance = balance - amount;
}
```

- **balance may not become exactly zero due to numerical inaccuracies (solution: use <=)**

```
while(power <= 1000){
    cout << "Next power of N is " << power << endl;
    power *= n;
}
```

- **Be sure to initialize to 0 a variable used for sums**
- **Be sure to initialize to 1 a variable used for products**

# Nested Loops

- Nested loops are loops within loops. They are similar in principle to nested `if` and `if-else` statements.
- Many applications require nested loops.

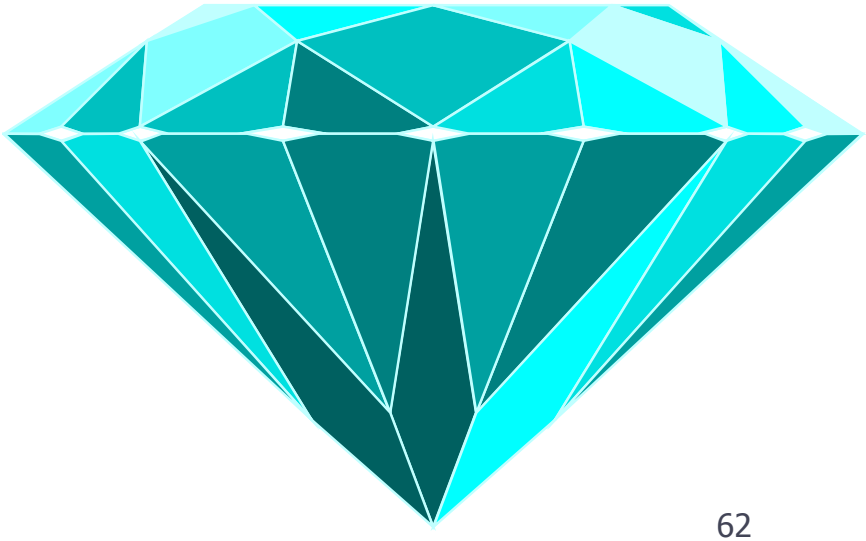
# Nested Loops

```
// Find the average score on 8 lab assignments
int n, lastlab=8;
double avg, score, tscore;
char resp;
do{
     tscore = 0;
     for(n=1; n<=lastlab; n++){
            cout << "Enter student's score for lab " << n <<
  ": ";
            cin >> score;
            tscore += score;
     }
     avg = tscore/double(lastlab);
     cout << "The average score is " << avg << endl;
     cout << "Enter another student (y/n)? ";
     cin >> resp;
}while(resp=='y' || resp=='Y');
```

# Diamond Pattern

- Print out the following diamond pattern

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
```

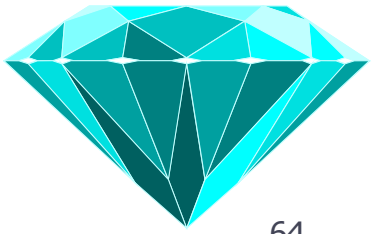


# Diamond Pattern

- Subproblem:
  - print out the upper half
  - print out the lower half
- Print out upper half:
  - row 1: print 4 spaces, 1 star;
  - row 2: print 3 spaces, 3 stars;
  - row 3: print 2 spaces, 5 stars;
  - row 4: print 1 space,  7 stars;
  - row 5: print 0 spaces, 9 stars;
- Algorithm Refinement:
  - row 1: print (5-row) spaces, (2*row - 1) stars;
  - row 2: print (5-row) spaces, (2*row - 1) stars;
  - row 3: print (5-row) spaces, (2*row - 1) stars;
  - row 4: print (5-row) spaces, (2*row - 1) stars;
  - row 5: print (5-row) spaces, (2*row - 1) stars;
  - i.e., row i: print (5-i) spaces, (2*i - 1) stars

# Diamond Pattern

```
int row, space, star;
 for(row=1; row<=5; row++){                    //top half
     for(space=1; space<=5-row; space++)
            cout << " ";
            for(star=1; star<=2*row-1; star++)
            cout << "*";
     cout << endl ;
}
for(row=4; row>=1; row--){                     //bottom half
     for(space=1; space<=5-row; space++)
            cout << " ";
     for(star=1; star<=2*row-1; star++)
            cout << "*";
     cout << endl ;
}
```



# Multiplication Table

```
// Program to output the
// multiplication table as a
// 2-dimensional table
int i;     //Outer loop counter
int j;     //Inner loop counter

for(i=1; i<=10; i++){
  for(j=1; j<=10; j++)
      cout << i*j << " ";
  cout << endl;
}
```