# Control Statements

Chapter 5, 6

If/Else, Loop

# Motivation

- Consider the Following Problems:
  - Write a program that allows the user to select whether the program will calculate the area of a triangle or of a rectangle
  - Write a program to print whether a steak is cooked to rare, medium, or done based on a temperature a user inputs
  - Modify your change program to detect if the user does not pay enough to cover what is owed
- All the above problems require a piece of code to be run only under certain conditions

# Control Statements

- A control statement controls which piece of code will be executed, and how many times it will be executed

- Control statements come in the form of:
  - Branches (If/Else)
  - Loops

- Control statements allow your program to make decisions

- Without control statements the same code would execute each time the program is run

# Boolean Expression

- Definition: An expression which can only evaluate to true or false

- Example:
  - $a < b$
  - In the event: a = 10, b = 15 the statement would be true
  - In the event: a = 15, b = 10 the statement would be false

- Example of Non-Boolean Expression:
  - $a * b$
  - The above expression will evaluate to a number and does not qualify as a Boolean expression (caveat explained later)

# Relational Operators

- All relational operators evaluate to a Boolean value
- These operators make a comparison between two operands
  - Equivalence: ==
  - Not Equivalent: !=
  - Less Than: <
  - Less Than or Equal To: <=
  - Greater Than: >
  - Greater Than or Equal to: >=

**Equivalence is not ‘=‘ (assignment) but using ‘=‘ instead of ‘==‘ will compile!!!!!!!**

# Example of Operators

int x = 5, y = 7, z = 5, a = 6;

- x == y
  - False
- x == z
  - True
- y < z
  - False
- y >= x
  - True

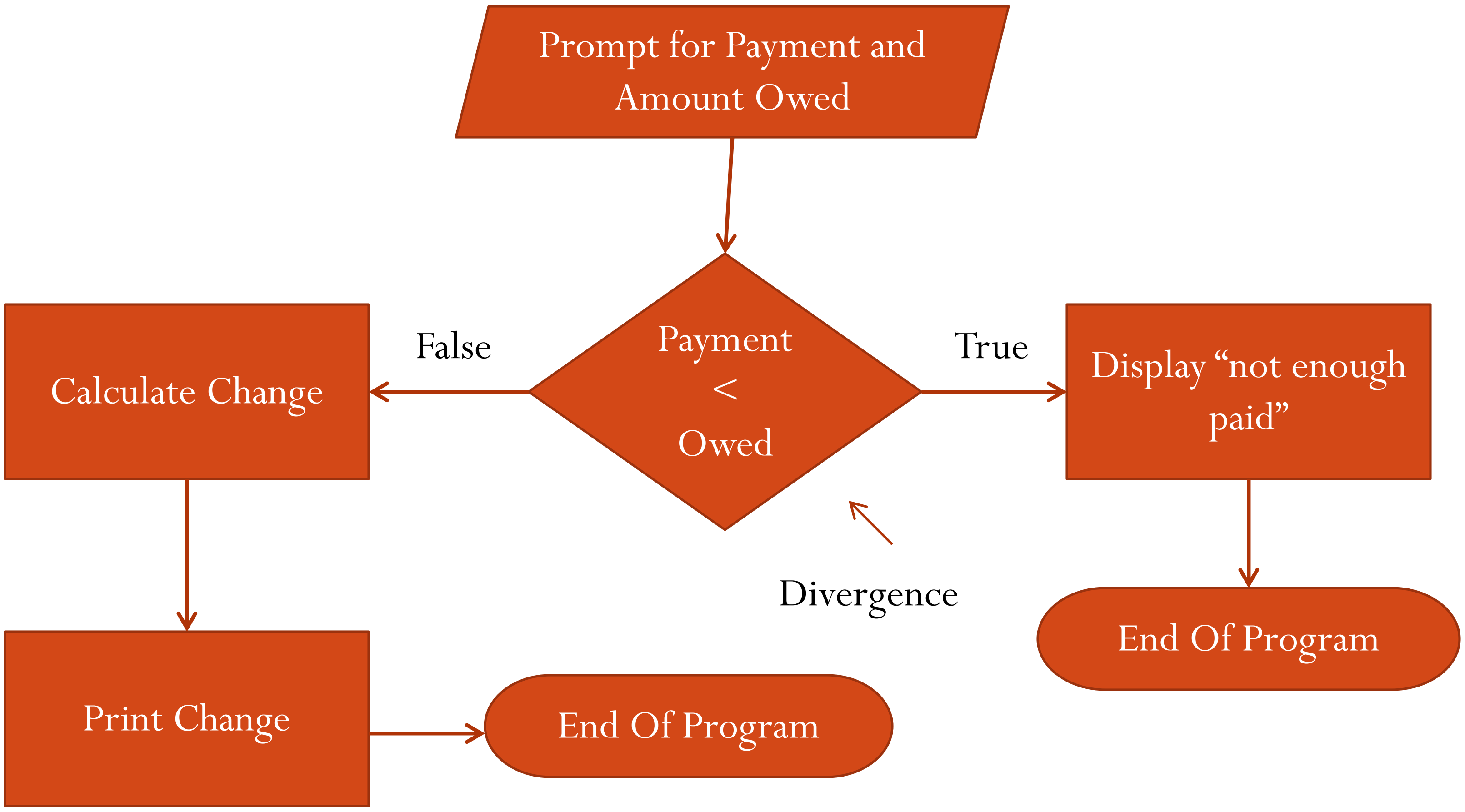
- x < a < y
  - Don’t Do This. You cant chain relational operators like in mathematics.
  - Will compile however

# Conditional Statement - If

```
if( condition )
{
        body of if
}
```

- The condition must evaluate to true or false
- If the condition evaluates to true then the body of the if is executed
- If the condition evaluates to false then the body of the if is skipped

# Flow Of Modified Change Program



# Modified Change Program

```
float payment, owed;
cout << "Enter payment and amount owed" << endl;
cin >> payment >> owed;
if(payment < owed)
{
        cout << "Not enough paid" << endl;
        return 0;
}

// Calculate and print denominations

return 0;
```

# If/Else

```
if( condition )
{
        body of if
}
else
{
        body of else
}
```

- The condition must evaluate to true or false
- If the condition evaluates to true then the body of the if is executed
- If the condition evaluates to false then the body of the else is executed

# Condition of the Else

```
if( x < 5)
{
        cout << "x is less than 5" << endl;
}
else
{
        cout << "x is greater or equal to 5" << endl;
}
```

- Never place a condition on an else, it is always implied
- if the condition is x < 5, then to be false the condition would be x >= 5
- x >= 5 is the implicit condition in which the else will be run

# Caveat: Non-Boolean Expressions

- In C++
  - zero is false
  - all non-zero numbers are true

- Therefore: a * b can be evaluated as a Boolean expression
- Example:
  - Lets suppose a = 2 and b = 3 then if( a * b) evaluates to true since 3 * 2 is 6 and 6 is a non-zero answer

- This is why '=', arithmetic operators, and chaining relational operators are syntactically correct in C++

# Non-Boolean Operator *if* statements

int x = 5, y = 7, z = 0, a;

- if(4 < y < 6)
  - 4 < y  evaluates to 1
  - Then 1 < 6 evaluates to true

- if(x * z)
  - x * z evaluates to 0, therefore false

- if(a = x * y)
  - x * y evaluates to 35
  - 35 is stored into a
  - a is non-zero, therefore true

# Guided Example 4.1

Write a program that allows the user to select whether the program will calculate the area of a triangle or of a rectangle

Write functions for the calculation of the area of the triangle and rectangle.

# Unguided Example 4.2

Write a program that allows the user to select whether the program will convert from Celsius to Fahrenheit, Celsius to Kelvin, or Fahrenheit to Celsius

Physically speaking you cannot go below -273.15 degrees Celsius or -459.67 Fahrenheit. Check the user's input to determine if the temperature is valid.

# Think about example

Write a program to print whether a steak is cooked to rare, medium, or done based on a temperature a user inputs

# Limitations of If

- Consider the following problems:
  - Modify a program that allows the user to run the program again without exiting
  - Convert a decimal  number to a n bit binary number
  - Calculate compound interest over n years
  - Calculate a Riemann sum
  - Wait a specified period of time
  - Write a function that calculates the power of a arbitrary base and exponent
- All of the above require code to be repeated for a number of times unknown at compile time

# Loops

```
while( condition )
{
        body of while
}
```

- Loops allow for code to be repeated while the condition is true
- Before each execution of the body, the condition is evaluated as a Boolean expression
- When the condition is true, the loop is executed

- Consider the condition as a mathematical constraint

# Short Hand Operators

- *var*++;
  - Increment Variable (i.e. *var* = *var* + 1;)
  - Example: i++;
- *var*--;
  - Decrement (i.e. *var* = *var* – 1;)
- *var* += *expresssion;*
  - Add a value to the variable (i.e. *var* = *var* + *expression*)
  - Example: x += 10;    adds 10 to x

- Additionally:  -=, *=, /=, %=

# Loop Example

```
int i = 0;
while( i < 5)
{
        cout << i << endl;
        i++;
}
```

Output:

0, 1, 2, 3, 4

- Loop iterates and increases *i* from 0 to 5
- When *i* reaches 5 the loop exits
- Since the condition fails when *i* is 5, i is not printed at that value

# Varied Loop Example

```
int i = 0;
while( i < 5)
{
        i++;
        cout << i << endl;
}
```

Output:

1, 2, 3, 4, 5

- Loop iterates and increases *i* from 0 to 5
- When *i* reaches 5 the loop exits
- Since the increment occurs prior to cout, the range is shifted by one
- Takeaway: Order of instructions is very important in loops
- General practice is to increment at the end of the loop

# Yet another variation

```
int i = 0;
while( i < 5)
{
        cout << i << endl;
}
```

Output:

0, 0, 0, ….. 0

- Loop iterates yet the value of *i* never changes
- If the values of *i* never changes then the stop condition never occurs
- Takeaway: Ensure the body of the loop modifies the variables in the condition such that an infinite loop does not occur

# Last variation, I promise

```
int i;
while( i < 5)
{
        cout << i << endl;
        i++;
}
```

Output:

?, ?+1, ?+2, ….. ????????

- *i* is uninitialized when entering the loop
- the loop will likely execute and exit the loop
- the results of the loop will be very unpredictable
- Takeaway: initialize the variable in your condition!!!

# Types of Loops

- Counting Loop
  - Contains a counting variable usually incrementing or decrementing by one
  - The number of times the loop is to be executed is known at *runtime* prior to entering the loop
- Sentinel Loop
  - Sentinel variable is tested for a condition to determine if to continue running
  - The number of executions of the loop isn't obvious prior to executing the loop

# Example of Counter Loop

```
int i = 0;
int n = 10;
while( i < n)
{
        cout << i << endl;
        i++;
}
```

- i is the counter variable
  - Increments by 1
- Condition checks the counter variable
- Prior to entering the loop it is obvious the loop will execute 10 times

# Example of Counter Loop

```
int i = 0;
int n = 10;
while( i < n)
{
        cout << i << endl;
        i++;
}
```

General Requirements of Counter Loop:

1. Initialize counter variable prior to executing the loop
2. Compare the counter variable to a threshold
3. Increment or decrement the counter variable inside the loop body (typically the last instruction)

# Exercise 4.3

- Write a program using loops that simulates compound interest. Prompt the user for the starting principle, interest rate, and the number of years of interest. Print the final result.

- Does the example require a loop?
  - Yes, the formula for compound interest must be applied in a repeated fashion for the specified number of years
- Is the example a counter loop?
  - Yes, the loop executions count up to the number of years specified by the user

# Sentinel Loop

```
int number;
cout << "Enter a number less than 100: ";
cin >> number;

while(number >= 100)
{
   cout << "Invalid Number." << endl;
   cout << "Enter a number less than 100: ";
   cin >> number;
}
```

- Sentinel variable is number
- Number of executions is not known prior to the execution of the loop
- Sentinel variable is compared to a condition

# General Rules for Sentinel Loops

```
int number;
cout << "Enter a number less than 100: ";
cin >> number;

while(number >= 100)
{
   cout << "Invalid Number." << endl;
   cout << "Enter a number less than 100: ";
   cin >> number;
}
```

1. Initialize sentinel variable prior to the loop
2. Compare sentinel variable to a value
3. Sentinel variable must have the ability to change inside the loop

# Example 4.4

- A forest has been recently planted on a barren plot of land. The number of trees that are initially planted is dictated by the user. The yearly reforestation rate for this particular tree type is 2%. If the initial number of trees is 250, then after year one the number of trees on the land is 250 * .02 + 250 yielding 255 trees. At the end of year two the number of trees can be calculated by 255 * .02 + 255. Write a loop that calculates the number of years it will take for the number of trees on the land to reach 1000 or above.

- Is this an example of a sentient loop?
  - Yes, the number of repetitions is dependent on the outcome of the calculations.
- What is the sentient variable?
  - The number of trees on the plot of land

# Approaching Loops

- How do I tell when to use a loop?
  - When a set of instructions are being repeated more than once
  - Remember: loops are for repetition, ifs are for decisions

- Designing Loops:
  1. Identify the type of loop
     - Counter versus Sentient
  2. Identify your starting conditions
     - these must be initialized prior to entering the loop
  3. Identify what is changing between loops
     - Is data accumulating?
     - Are you counting?
  4. Identify your stop condition
     - usually based on the data changing in the loops
     - to find the condition, just take the inverse of the stop condition

# Guided Exercise 4.3

- Write a program using loops that simulates compound interest. Prompt the user for the starting principle, interest rate, and the number of years of interest. Print the final result.

- Designing the loop
  - Identify the type of loop
    - Counter loop
  - Identify what is changing between loops
    - counting years until stop year
  - Identify your starting conditions
    - principle, interest rate, and years are set by user
    - current year must be set to 0
  - Identify your stop condition
    - when current year is greater or equal to stop year
    - opposite is, while current year is less than the stop year

# Guided Exercise 4.5

- Write a program to convert a decimal number of any value inputted by the user to a binary number.

# Unguided Exercise 4.6

- Write a program to convert a binary number of any value inputted by the user to a decimal number.