# Control Flow

## © 2005 Devendra Tewari

# if-else

## if-else expresses decisions

## else part is optional

### if (*expression*)

### *statement*1

### else

### *statement*2

## *expression* returns a numerical value, 0 is considered FALSE, any other value is TRUE

# if-else ambiguities

## if (i >= 0)

## if (i < 5)

## a = b;

## else

## a = c;

## By default the else is associated with the inner if

## if (i >= 0) {

## if (i < 5)

## a = b;

## } else

## a = c;

## Use braces to remove ambiguity

# else-if

## Useful for expressing multi-way decisions

### if (*expressions*)

### *statement*

### else if (*expression*)

### *statement*

### else if (*expression*)

### *statement*

### else

### *statement*

# switch

## Used to express multi-way decision

## Matches the result of an expression to one of several integer constants

### switch (*expression*) {

### case *const-expr*: *statements*

### case *const-expr*: *statements*

### default: *statements*

### }

## a break statement causes exit from the switch, without a break all statements after the matching case are executed till the end of the switch block

# while

## while (*expression*) {

## *statements*

## }

## The while loop executes as long as *expression* is TRUE (not 0)

# for

## for (*expr*1; *expr*2; *expr*3) {

## *statements*

## }

## The loop has three parts, *expr*1 is an initialization expression, *expr*2 is a relational expression and *expr*3 is the increment expression

## The loop executes as long as *expr*2 is TRUE

## All three expressions can be empty which leads to an infinite for loop

# do-while

## do {

## *statements*

## } while (*expression*);

## The do loop executes at least once before *expression* is evaluated

## The loop executes as long as *expression* is TRUE

# break

## Using the break statement causes immediate exit from a loop (for, while or do-while) or switch block

# continue

## The continue statement causes a loop to begin the next iteration, the statements following continue are not executed

# goto

## goto *label*;

## *statements*

## *label*:

## *statements*

## the goto statement causes execution to jump to the statements after the *label*

## goto is not recommended as it results in spaghetti code

# Exercise

## Write a program that converts 1 to 50 mile(s) into kilometers.

### NOTE: 1 mile = 1.609344 kilometers

## Print the result in tabular form as shown below