UEE1302 Introduction to Computers and Programming

C_Lecture 02:

Control Statements: Part I

(if, if … else, and while)

C: How to Program 7th ed.

Agenda

- Relational Expressions and Operators
- `if-else` Statement
- Nested `if` and `if` chain statement
- `while` Statement
- Counter-controlled Repetition and Sentinel-controlled Repetition
- Increment and Decrement Operators

Algorithm

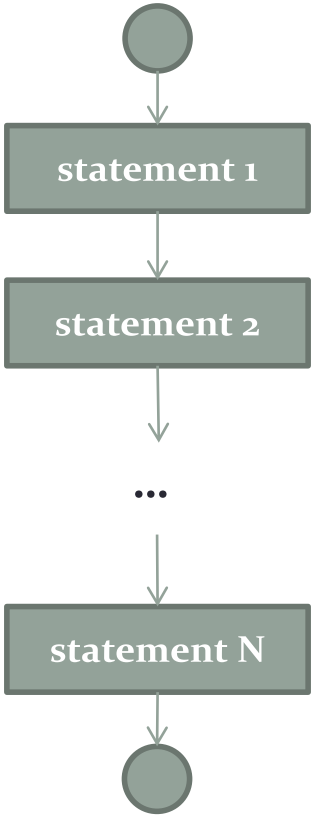
- Any solvable computing problem can be solved by the execution of a series of actions in a specific order
- An algorithm is procedure for solving a problem in terms of
  - the actions to execute and
  - the order in which the actions execute
- Specifying the order in which statements (actions) execute in a computer program is called program control
- This chapter investigates program control using C's control statements

Control Structures

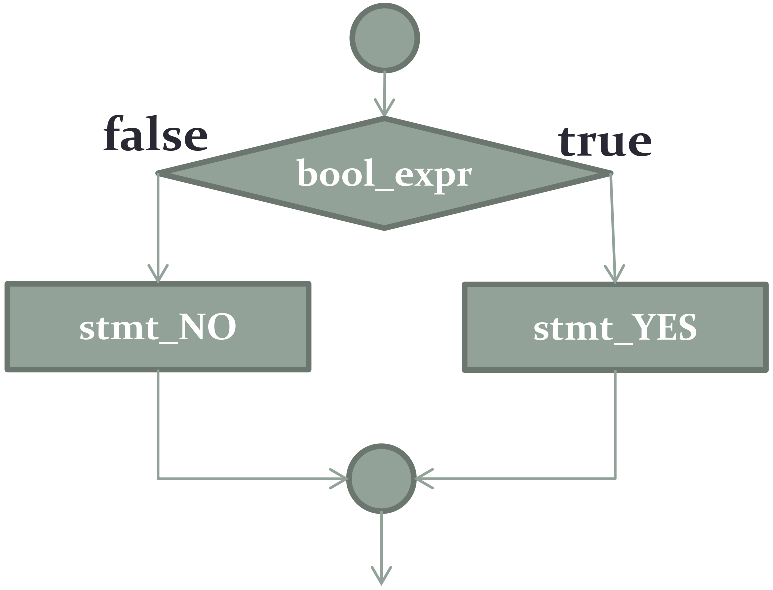
- Flow of Control: the order in which a program's statements are executed
    - Normal flow is sequential

- Selection and Repetition statements allow programmer to alter normal flow
    - Selection: selects a particular statement to be executed next => selection is from a well-defined set
    - Repetition: allows a set of statements to be repeated

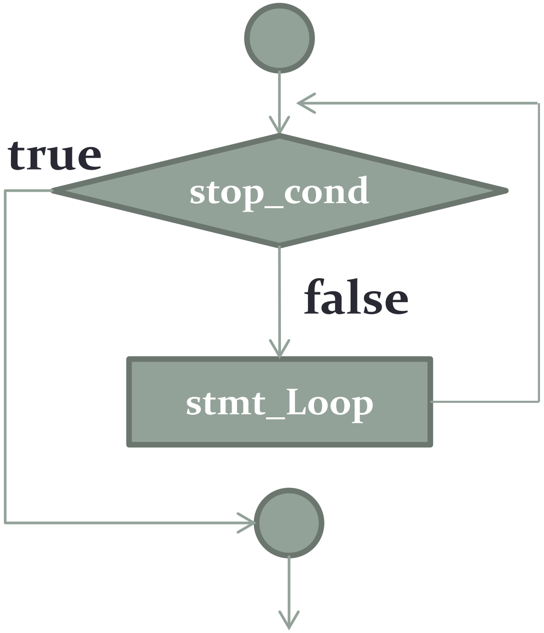
Flow of Execution



(1) Sequence



(2) Selection



(3) Repetition

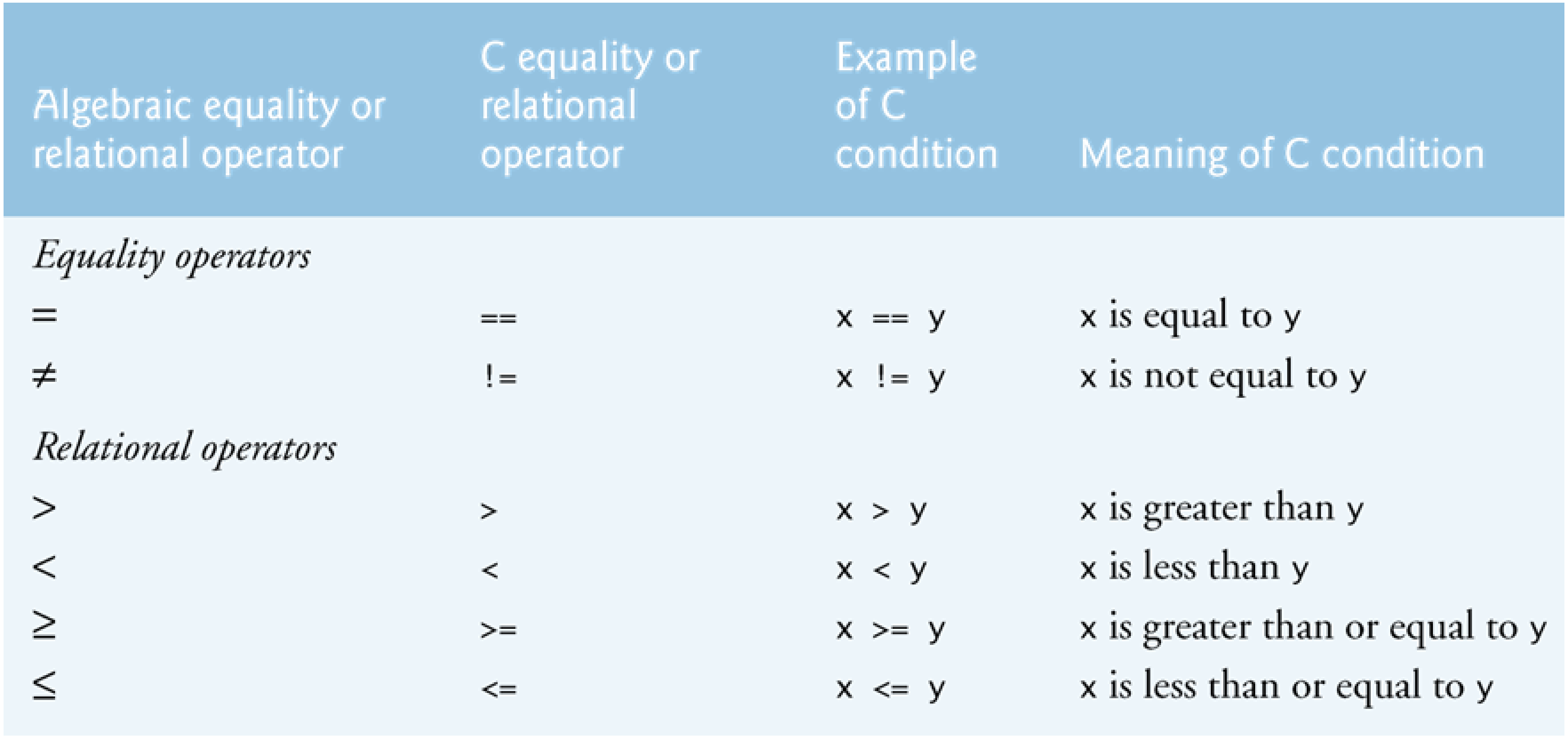
Relational Expressions

- All computers are able to compare numbers
  - Can be used to create an intelligence-like facility
- Relational Expressions: expressions are used to compare operands
  - Format: a relational operator connecting two variable and/or constant operands
  - Examples of valid relational expressions:
    - `Age > 40`
    - `length <= 50`
    - `flag == done`

Relational Expressions (cont.)

- Relational Expressions (conditions) are evaluated to yield a numerical result
  - If condition is true, result becomes 1
  - If condition is false, result becomes 0
  - Example:
    - The relationship `2.0 > 3.3` is always false, therefore the expression has a value of 0

Equality and Relational Operators



Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Flow of Control: Selection

- `if/else` statement

```
if (score >= 90) {
    grade = 'A';
} else if (score >= 80) {
    grade = 'B';
} else {
    grade = 'C';
}
```

- Ternary operator

```
(grade > 60)? printf("pass"): printf("fail");
```

Flow of Control: Selection (cont.)

- switch statement (in Chapter 4)

```
switch (menu) {
    case item1:
        //do something
        break;
    case item2:
    case item3:
        //do something
        break;
    ...
    default:
        //do something
        break;
}
```

Flow of Control: Selection (cont.)

- The `if` selection statement is a single-selection statement because it selects or ignores a single action (or, as we'll soon see, a single group of actions)
- The `if`…`else` statement is called a double-selection statement because it selects between two different actions (or groups of actions)
- The `switch` selection statement is called a multiple-selection statement because it selects among many different actions (or groups of actions)

Selection (I): One-Way `if`

- Formal syntax of one-way selection :

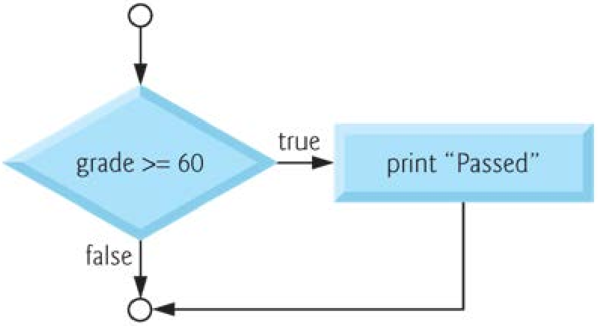
```
…
if ( decision_maker ) //no ; here
    action_stmt;
…
```

  - decision_maker: is a logical expression decides whether to execute the action statement
  - if decision_maker is true, execute action_stmt
  - if decision_maker is false, bypass action_stmt

Examples of One-Way `if`

- Example 1:

```
if ( grade >= 60 )
    printf("Passed");
```



- Example 2: absolute value

```
…
if ( iVar < 0 )
    iVar = -iVar;
printf("absolute value = %d\n", iVar);
…
```

Selection (I): Two-Way `if`

- Choice of two alternate statements based on condition expression
- Formal syntax :

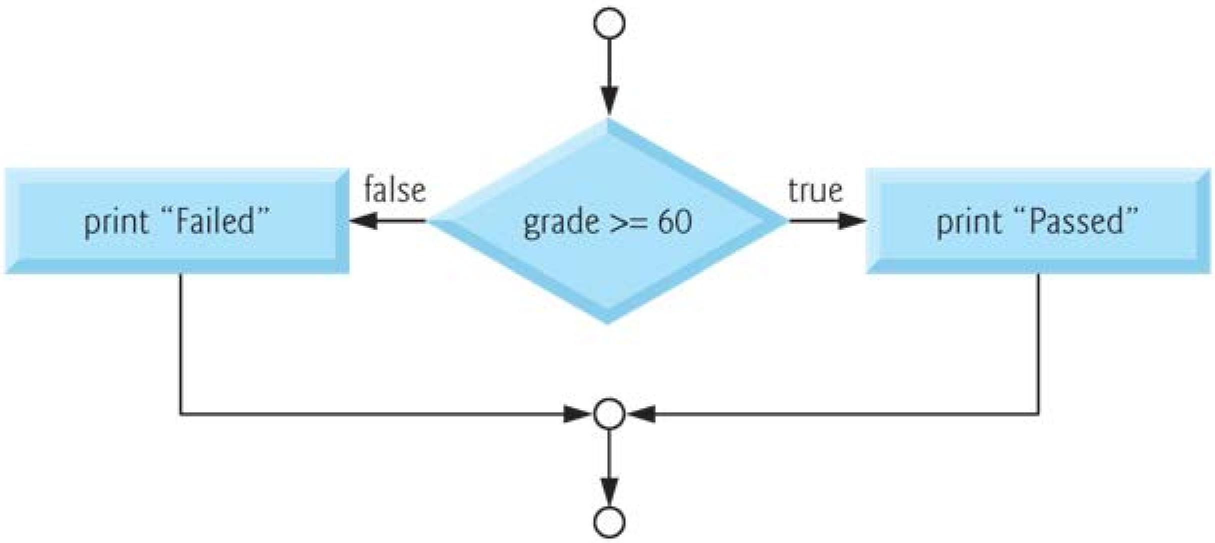
```
if ( decision_maker ) // no semicolon here
    action_stmt_yes;
else                  // no semicolon here
    action_stmt_no;
```

  - decision_maker:
    - decide which one of two statements to run
  - if the evaluation is true, run action_stmt_yes
  - if the evaluation is false, run action_stmt_no

Examples of Two-Way `if`

- Example 1: pass or fail

```
    if ( grade >= 60 )
        printf("Passed");
    else
        printf("Failed");
```



Examples of Two-Way `if`

- Example 2: overtime payment

```
…
if ( hours > 40 )
    pay = rate*40 + 1.5*rate*(hours-40);
else
    pay = rate*hours;
…
```

Compound Statement

- What if we want to execute multiple statements in action?
- Compound statement (a.k.a. a block of statements):

```
{
    statement 1;
    statement 2;
    …
    statement N;
}
```

  - a compound statement is treated as a single statement

Example of Compound Statements

```
if ( grade >= 60 )
    printf("Passed");
else {
    printf("Failed.\n");
    printf("Take this courser again");
}
```

Nested `if`

- Nesting: one control statement in another
- An else is associated with the most recent if that has not been paired with an else

```
if ( grade >= 90 )
    printf("A");
else
    if ( grade >= 80 )
        printf("B");
    else
        if ( grade >= 70 )
            printf("C");
        else
            if ( grade >= 60 )
                printf("D");
            else
                printf("F");
```

Avoid Excessive Indentation

- Use `if` chain instead of nested `if`

```c
if ( grade >= 90 )
    printf("A");
else if ( grade >= 80 )
    printf("B");
else if ( grade >= 70 )
    printf("C");
else if ( grade >= 60 )
    printf("D");
else
    printf("F");
```

Compare `if` Chain and Multiple `if`

- `if` Chain

```
if ( month == 1 )
    printf("Jan.\n");
else if ( month == 2 )
    printf("Feb.\n");
else if ( month == 3 )
    printf("Mar.\n");
…
else if ( month == 11 )
    printf("Nov.\n");
else
    printf("Dec.\n");
```

- multiple `if`

```
if ( month == 1 )
    printf("Jan.\n");
if ( month == 2 )
    printf("Feb.\n");
if ( month == 3 )
    printf("Mar.\n");
…
if ( month == 11 )
    printf("Nov.\n");
if ( month == 12 )
    printf("Dec.\n");
```

Question: What’s the difference??

Common Pitfalls on `if-else`

- Operator "`=`" vs. operator "`==`"
  - "assignment“ versus “equality"
- Example: What’s the problem??

```
if ( age = 20 )
    printf("Happy 20-year old birthday.\n");
else
     …
```

  - Using “=” instead of ”==” in the if-else statement causes the most difficult errors
    - Hard to debug due to no error message

Dangling-else Problem

- The C compiler always associates an else with the immediately preceding if unless told to do otherwise by the placement of braces ({ and }).
- This behavior can lead to what’s referred to as the dangling-else problem.

```
if ( x > 5 )
   if ( y > 5 )
      printf("x and y are > 5");
else
   printf("x is <= 5");
```

Dangling-else Problem

- The compiler actually interprets the statement as

```
if ( x > 5 )
   if ( y > 5 )
      printf("x and y are > 5");
   else
      printf("x is <= 5");
```

- To force the nested if-else statement to execute as intended, use:

```
if ( x > 5 ) {
   if ( y > 5 )
      printf("x and y are > 5");
}
else
   printf("x is <= 5");
```

Selection (II): Conditional operator ? :

- Conditional operator (? :) takes three arguments (ternary)
  - equivalent to `if-else`
- Syntax for the conditional operator:

```
var = expr_1 ? expr_2 : expr_3;
```

  - if expr_1 is true, assign expr_2 to var
  - if expr_1 is false, assign expr_3 to var
- Example

```
(grade > 60)? printf("pass"): printf("fail");
```

```
printf("%s\n", grade > 60 ? "pass": "fail");
```

Why is Repetition Needed?

- Repetition (a.k.a. Loop)
  - allow you to efficiently use variables
  - can input, add, and average multiple numbers using a limited number of variables
- For example, to add five numbers:
  - declare a variable for each number, input the numbers and add the variables together
  - create a loop that reads a number into a variable and adds it to a variable that contains the sum of the numbers

Flow of Control: Repetition

- while loop

```
while ( i < 5 ) {
    printf("good!\n");
    i++;
}
```

- do/while loop **(in Chapter 4)**

```
do {
    printf("good!\n");
    i++;
} while ( i < 5 );
```

Flow of Control: Repetition (cont.)

- for loop (in Chapter 4)

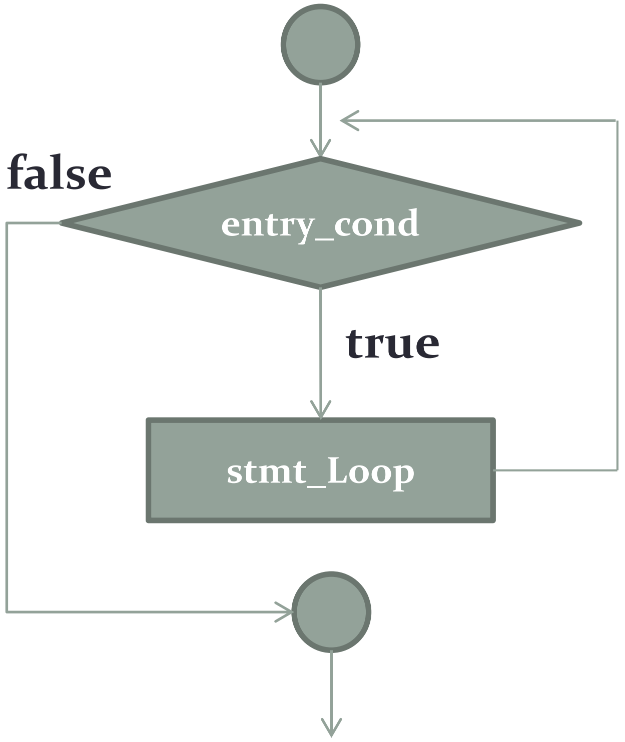
```
for ( i = 0; i < 5; i++ ) {
    printf("good!\n");
}
```

Repetition (I) : while loop

- Syntax of the while statement is:

```
while (entry_cond)
   stmt_Loop;
```

  - Statement stmt_Loop can be either simple or compound => body of the loop
  - Expression entry_cond acts as a decision maker and is usually a Boolean expression



Repetition (I) : `while` loop (cont.)

- Expression `entry_cond` provides an entry condition
- Statement `stmt_Loop` executes if the expression initially evaluates to true
- Loop condition `entry_cond` is then reevaluated
- Statement continues to execute until the expression `entry_cond` is no longer true
- Infinite loop: continues to execute endlessly
    - can be avoided by including statements in the loop body that assure exit condition will eventually be true

Example of `while` loop

```c
int product  = 3;

while ( product < 100 ) {
    product = 3 * product;
    printf("%d\n", product);
}
```

- screen output:

```
9
27
81
243
```

`while` (1): Counter-Controlled

- If you know exactly how many pieces of data need to be read, the `while` loop becomes a counter-controlled loop
- Example of counter-controlled loop:

```
counter = 0; // initialize loop control
while ( counter < Limit ){ // test loop control
    statements;
    …
    counter++; // update loop control
}
```

`while` (2): Sentinel-Controlled

- Sentinel variable is tested in the condition and loop ends when sentinel is encountered
- Example of sentinel-controlled loop:

```
scanf("%d", &target); // initialize loop control
//test loop control
while ( target != sentinel ) {
    statements;
    // update loop control
    scanf("%d", &target);
}
```

Example of Counter-Controlled Loop

```
// Modify from Fig. 3.6: fig03_06.c
// class average program with counter-controlled
repetition
#include <stdio.h>

int main ( void )
{
    // declaration and initialization
    // number of grade to be entered next
    unsigned int counter;
    int grade = 0; // grade value
    int total; // sum of grades entered by user
    int average = 0; // average of sum

    // initialization
    total = 0;
    counter = 1;
```

Example of Counter-Controlled Loop (cont.)

```
    // processing
    while ( counter <= 10 ) { // loop 10 times
        printf("Enter grade: ");
        scanf("%d", &grade); // input grade
        total = total + grade; // add grade to total
        counter = counter + 1; // increment
    } // end while

    average = total / 10;
    printf("\nTotal of all 10 grade is %d\n", total);
    printf("Class average is %d\n", average);

    return 0;
}
```

Example of Counter-Controlled Loop (cont.)

- screen output

```
Enter grade: 67
Enter grade: 78
Enter grade: 89
Enter grade: 67
Enter grade: 87
Enter grade: 98
Enter grade: 93
Enter grade: 85
Enter grade: 82
Enter grade: 100

Total of all 10 grades is 846
Class average is 84
```

Example of Sentinel-Controlled Loop

```
// Modify from Fig. 3.8: fig03_8.c
// Class average program with sentinel-controlled
repetition
#include <stdio.h>

int main ( void )
{
    // declaration and initialization
    unsigned int counter = 0; // number of grades entered
    int grade = 0; // grade value
    int total = 0; // sum of grades entered by user
    float average = 0; // average of sum

    printf("Enter grade, -1 to end: ");
    scacnf("%d", &grade); // read grade from user
```

Example of Sentinel-Controlled Loop (cont.)

```
    while ( grade != -1 ) { // while grade is not -1
        total = total + grade; // add grade to total
        counter = counter + 1; // increment
        printf("Enter grade, -1 to end: ");
        scacnf("%d", &grade); // read grade from user
    } // end while
    if ( counter != 0 ) {
        average = (float) total / counter;
        printf("\nTotal of all %d grades entered is %d",
               counter, total);
        printf("Class average is %.2f\n", average);
    }
    else
        printf("No grades were entered.\n");
    return 0;
}
```

Example of Sentinel-Controlled Loop (cont.)

- screen output

```
Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of all 3 grades is 257
Class average is 85.67
```

```
Enter grade or -1 to quit: -1

No grades were entered
```

`while` Pitfalls: Misplaced ;

- Watch the misplaced ; (semicolon)
- Example

```
while ( response != 0 );
{
    printf("Enter val: ");
    scanf("%d", &response);
}
```

  - Notice the "`;`" after the while condition!
  - Result here: INFINITE LOOP!

`while` Pitfalls: Infinite Loops

- Loop condition must evaluate to false at some iteration through loop
  - If not => infinite loop
- Example

```
while ( 1 )
{
    printf("Hello ");
}
```

  - a perfectly legal C loop => always infinite!
- Sometimes, infinite loops can be desirable
  - e.g., "Embedded Systems"

Problem Statements

- A college offers a course that prepares students for the state licensing exam for real estate brokers. Last year, 10 of the students who completed this course took the licensing examination. Naturally, the college wants to know how well its students did on the exam. You've been asked to write a program to summarize the results. You've been given a list of these 10 students. Next to each name a 1 is written if the student passed the exam and a 2 if the student failed.
  - Input each test result (i.e., a 1 or a 2). Display the prompting message "Enter result" each time the program requests another test result.
  - Count the number of test results of each type.
  - Display a summary of the test results indicating the number of students who passed and the number who failed.
  - If more than eight students passed the exam, print the message "Bonus to instructor!"

Example of Nested Control Structures

```
// Fig. 3.10: fig03_10.c
// Analysis of examination results
#include <stdio.h>

int main ( void )
{
    // declaration and initialization
    unsigned int passes = 0;  // number of passes
    unsigned int failures = 0; // number of failures
    unsigned int student = 1; // student counter
    int result;    // one exam result
```

Example of Sentinel-Controlled Loop (cont.)

```
    while ( student <= 10 ) { // process 10 students
        printf( "Enter result (1=pass, 2=fail): ");
        scanf("%d", &result);

        if ( result == 1 )
            passes = passes + 1;
        else
            failures = failures + 1;

        student = student + 1;
    } // end while

    printf("Passes %d\n", passes);
    printf("Failed %d\n", failures);

    if ( passes > 8 )
        printf("Bonus to instructor!\n");
    return 0;
}
```

Example of Sentinel-Controlled Loop (cont.)

- screen output

```
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 2
Enter Result (1=pass, 2=fail): 2
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 2
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 1
Enter Result (1=pass, 2=fail): 2
Passed 6
Failed 4
```

Shorthand Operators

- Shortcut Assignment Operator
  - `+=      -=      *=      /=      %=`
- Example:

```
sum = sum + 10;
```

can be written as

```
sum += 10;
```

Shorthand Operators (cont.)

- Increment Operators (`++`) and Decrement Operator (`- -`)
  - Unary operator for special case when variable is increased or decreased by 1
  - Using the increment operator, the expression

    ```
    variable = variable + 1;
    ```

    can be replaced by either

    ```
    ++variable;
    ```

    or

    ```
    variable++;
    ```

Post-Increment vs. Pre-Increment

- Prefix increment operator: the `++` or `--` operator appears before a variable
  - The expression `k = ++n` does two things

```
n = n + 1; // increment n first
k = n; // assign n’s value to k
```

- Postfix increment operator: the `++` or `--` operator appears after a variable
  - The expression `k = n++` works differently

```
k = n; // assign n’s value to k
n = n + 1; // and then increment n
```

Example of Post/Pre-Increment

```c
// Fig. 3.13: fig03_13.c
// preincrementing and postincrementing
#include <stdio.h>

int main ( void )
{
    int c;
    c = 5;  // assign 5 to c
    printf("%d\n", c);    // print 5
    printf("%d\n", c++);  // print 5 then postincreament
    printf("%d\n\n", c);  // print 6
```

Example of Post/Pre-Increment (cont.)

```
    c = 5;
    printf("%d\n", c);    // print 5
    printf("%d\n", ++c);  // preincrement then print 6
    printf("%d\n", c);    // print 6

    return 0;
}
```

- screen output:

```
5
5
6

5
6
6
```

Summary

- `if-else` statements select between two alternative statements based on the value of an expression
- `if-else` statements can contain other `if-else` statements => nested `if`
  - If braces are not used, each `else` statement is associated with the closest unpaired `if`
- `if` chain: a multi-way selection statement
  - Each `else` statement (except for the final else) is another `if-else` statement
- Compound statement: any # of individual statements enclosed within braces {}

Summary (cont.)

- `while`: expression is the decision maker, and the statement is the body of the loop
- In a counter-controlled while loop,
  - Initialize counter before loop
  - Body must contain a statement that changes the value of the counter variable
- A sentinel-controlled while loop uses a sentinel to control the while loop

Exercise (1)

- Write single C statements that
  a) Input unsigned integer variable `x` with `scanf`. Use the conversion specifier `%u`.
  b) Input unsigned integer variable `y` with `scanf`. Use the conversion specifier `%u`.
  c) Set unsigned integer variable `i` to 1.
  d) Set unsigned integer variable `power` to 1.
  e) Multiply unsigned integer variable `power` by `x` and assign the result to `power`.
  f) Increment variable `i` by 1.
  g) Test `i` to see if it's less than or equal to y in the condition of a `while` statement.
  h) Output unsigned integer variable `power` with `printf`. Use the conversion specifier `%u`.

Exercise (2)

- Write a C program that uses the statements in Exercise (1) to calculate x raised to the y power. The program should have a while repetition control statement.

Exercise (3)

- What's wrong with the following while repetition statement (assume `z` has value 100), which is supposed to calculate the sum of the integers from 100 down to 1?

```
while ( z >= 100 )
    sum += z;
```