Recap to previous lecture!

- Switch cases
- While loop
The **switch** Statement

- The *switch* statement is an alternative:

```c
int grade = 3;
switch (grade) {
    case 4: printf("Excellent"); break;
    case 3: printf("Good"); break;
    case 2: printf("Average"); break;
    case 1: printf("Poor"); break;
    case 0: printf("Failing"); break;
    default: printf("Illegal grade"); break;
}
```
The `switch` Statement

• The word `switch` must be followed by an `integer expression`—the `controlling expression`—in parentheses.

• *Characters* are treated as integers in C and thus can be tested in `switch` statements.

• *Floating-point numbers and strings* don’t qualify, however.
Loops in C
• C provides three loop statements:
  – The **while statement** is used for loops whose controlling expression is tested **before** the loop body is executed.
  – The **do statement** is used if the expression is tested **after** the loop body is executed.
  – The **for statement** is convenient for loops that increment or decrement **a counting variable**.
The **while** Statement

- Using a **while** statement is the easiest way to set up a loop.
- The **while** statement has the form
  
  ```
  while ( expression )
  {
    statement
  }
  ```

- *expression* is the controlling expression; *statement* is the loop body.
• Develop C program that prints countdown messages from 10 to 0, in separate lines.

```c
int i = 10;
while (i > 0) {
    printf(" %d \n", i);
    i--;
}
printf(" Done ! \n");
```
i = 10;
while (i > 0) {
    printf(" %d \n", i);
    i++;
}
printf(" done with the loop");
Infinite Loops

• A while statement won’t terminate if the controlling expression *always has a nonzero value*.

• C programmers sometimes deliberately create an *infinite loop* by using a *nonzero constant* as the controlling expression:

  ```
  while (1) ...
  ```

• A while loop of this form will execute forever unless its body contains a *statement that transfers control out of the loop* (e.g., `break`) or calls a function that causes the program to terminate.
Exiting from a Loop

• The normal exit point for a loop is at the beginning (as in a `while` or `for` statement) or at the end (the `do` statement).

• Using the `break` statement, it’s possible to write a loop with an exit point *in the middle or a loop with more than one exit point*. 
The `break` Statement

- The `break` statement can transfer control out of a switch statement, but it can also be used to *jump out* of a `while`, `do`, or `for` loop.

- Checks if number `num` *is prime*, use a `break` statement to terminate the loop as soon as a divisor is found.

```c
int d = 2;
int num = 100;
while ( d < num ){
    if (num % d == 0)
        break;
    d++;
}
```
The **break** Statement

- A **break** statement transfers control *out of the innermost enclosing while, do, for, or switch*.
- Example:

```java
while (...) {
    switch (...) {
        ...
        break;
        ...
    }
}
```

- **break** transfers control *out of the switch statement, but not out of the while loop*. 
The continue Statement

• The `continue` statement is similar to `break`:
  – `break` transfers control just past the end of a loop.
  – `continue` transfers control to a point just before the end of the loop body.

• `break` leaves the loop while `continue` leaves the current iteration of the loop.
A loop that uses the `continue` statement:

```c
n = 0;
sum = 0;
while (n < 10) {
    scanf("%d", &i);
    if (i == 0)
        continue;
    sum += i;
    n++;
    /* continue jumps to here */
}
```
- The same loop written without using `continue`:

```c
n = 0;
sum = 0;
while (n < 10) {
    scanf("%d", &i);
    if (i != 0) {
        sum += i;
        n++;
    }
}
```
do .. while, loop

• General form of the do statement:

```c
do {
    statements
} while ( expression );
```

• When a do statement is executed, the loop body is executed first, then the controlling expression is evaluated.

• If the value of the expression is true, the loop body is executed again and then the expression is evaluated once more.
• The countdown example rewritten as a do statement:

```c
int i = 10;
do {
    printf("T minus %d and counting\n", i);
    --i;
} while (i > 0);
```

• The do statement is often indistinguishable from the while statement, however the body of a do statement is always executed at least once.
The `for` Statement

- The `for` statement is ideal for loops with "counting" variable.
- General form of the `for` statement:

```plaintext
for ( expr1 ; expr2 ; expr3 )
{
    statements
}
```

`expr1`, `expr2`, and `expr3` are expressions.
for ( expr1; expr2; expr3 )
{
    statements
}

- `expr1` is an *initialization step* that’s performed only once, before the loop begins to execute.
- `expr2` *controls loop termination* (the loop continues executing as long as the value of `expr2` is nonzero).
- `expr3` is an operation to be performed at the end of *each loop iteration*. 
The **for** Statement

```c
int i;
for ( i = 0 ; i < 10 ; i++ ) {
    printf("value of i is %d \n", i);
}
```

```c
int j=0;
while( j < 10 ) {
    printf("value of j is %d \n", j);
    j++;
}
```
for Statement Idioms

- The for statement is usually the best choice for loops that “count up” (increment a variable) or “count down” (decrement a variable).

**Counting up from 0 to n–1:** \( \text{for } (i = 0; i < n; i++) \)

**Counting up from 1 to n:** \( \text{for } (i = 1; i <= n; i++) \)

**Counting down from n–1 to 0:** \( \text{for } (i = n - 1; i >= 0; i--) \)

**Counting down from n to 1:** \( \text{for } (i = n; i > 0; i--) \)
Program: Printing a Table of Squares (Revisited)

• Utilize for loop to print the table of squares, from 1 up to n (user input).
#include <stdio.h>

int main(void)
{
    int i, n;

    printf("Enter number of entries: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
        printf("%10d%10d
", i, i * i);

    return 0;
}
Program: Summing a Series of Numbers

• Develop a program that sums a series of integers entered by the user:

  This program sums a series of integers. Enter integers (0 to terminate): 8 23 71 5 0
  The sum is: 107
#include <stdio.h>

int main(void)
{
    int n, sum = 0;

    printf("This program sums a series of integers.\n");
    printf("Enter integers (0 to terminate): ");

    scanf("%d", &n);
    while (n != 0) {
        sum += n;
        scanf("%d", &n);
    }
    printf("The sum is: %d\n", sum);

    return 0;
}
• Write a program that finds the largest in a series of numbers entered by the user, the program must prompt the user input one by one. When the user enters 0, the program must display the largest number entered.

Enter a number: 60
Enter a number: 100.43
Enter a number: 75.234
Enter a number: 0
The largest number entered was 100.43
• Write a program that prompts the user to enter a number $n$, then prints all even squares between 1 and n. For example, if the user enters 100, the program should print:

4
16
36
64
100
• Develop a program that counts number of digits in an integer, using while loop.