C Programming

Recitation 3

March 2016

Outline

Switch Statement
- Switch Statement
- Multiple Cases

Loop Statements
- A Simple Loop
- Loop Statements
- The *while* loop
- The *for* loop
- The *for* loop
- The *do – while* loop
- Nested Loops

Loop Interruption
- *break* Statement
- *continue* Statement

Control Flow
- goto

Functions
- Functions

SWITCH STATEMENT

- Alternative conditional statement
- Integer (or character) variable as input

```
switch (ch) {
        case 'Y': /* ch == Y */
                /* do something */
                break;
        case 'N': /* ch == N */
                /* do something else */
                break;
        default: /* otherwise */
                /* do a third thing */
                break;
}
```

Multiple Cases

- Compares variable to each case in order
- When match found, starts executing inner code until *break*; reached

```
switch (ch) {
        case 'Y':
                /* do something if ch == 'Y' */
        case 'N':
                /* do something if ch == 'Y' or ch == 'N' */
                break;
}
```

A SIMPLE LOOP

```
/* finding the sum of the first n terms of the series 1/1 + 1/2 +↩
      1/3 + ... */
#include <stdio.h>
int main() {
      int i, n;
      float sum;
      scanf("%d", &n);
      i = 1, sum = 0;
      while (i <= n) {
            sum += 1.0/i;
            i++;
      }
      printf("sum = %f\n", sum);
      return 0;
}
```

Loop Statements

- The *while* loop
- The *for* loop
- The *do − while* loop
- The *break* and *continue* keywords

THE *while* LOOP

```
while ( /* condition */ )
      /* loop body */
```

- Simplest loop structure - evaluate body as long as condition is true
- Condition evaluated first, so body may never be executed
- *Example 1:* Write a program that determines what fraction of a given text consists of vowels
- *Example 2:* Take two integers as input and print the greatest common divisor of them

i	j	result(i,j)
119	35	84
84	35	49
49	35	14
14	35	21
14	21	7
14	7	7
7	7	–

THE *for* LOOP

```
int factorial(int n) {
    int i, j = 1;
    for (i = 1; i <= n; i++)
        j = j * i;
    return j;
}
```

- The "counting" loop
- Inside parentheses, three expressions, separated by semicolons:
  - Initialization: $i = 1$
  - Condition: $i <= n$
  - Increment: i++
- Expressions can be empty (condition assumed to be "true")
- *Example 3:* Read 10 integers and print average, maximum and minimum of those numbers

The *for* loop

```
int factorial(int n) {
    int i, j;
    for (i = 1, j = 1; i <= n; j = j * i, i++)
        ;
    return j ;
}
```

- Compound expressions separated by commas
- Comma: operator with lowest precedence, evaluated left-to-right

THE *do – while* LOOP

```
char c;
do {
    /* loop body */
    puts("Keep going? (y/n) ");
    c = getchar ();
    /* other processing */
} while (c == 'y' && /* other conditions */ );
```

- Differs from *while* loop – condition evaluated after each iteration
- Body executed at least once
- Note semicolon at end
- *Example 4:* Determine the number of digits and their sum in a nonnegative decimal integer

Nested Loops

- A loop may contain other loops within its body
- *Example 5:* Draw a pyramide
- *Example 6:* Three positive integers a, b, and c, such that $a < b < c$, form a special triplet if $a^2 + b^2 = c^2$. Write a program that generates all these triplets a, b, c where a, b <= 25

break STATEMENT

- Sometimes want to terminate a loop early
- *break*; exits innermost loop or switch statement to exit early
- Consider the modification of the *do – while* example:
- *Example:* purpose of this code block?

```
for (i = 1; i <= 100; i++) {
        for (j = 2; j <= sqrt(i); j++) {
                if (i % j == 0)
                        break;
        }
        if (j > sqrt(i))
                sum += i;
}
```

continue STATEMENT

- Use to skip an iteration
- *continue*; skips rest of innermost loop body, jumping to loop condition
- *Example:* Determine the sum of integers from 1 to n, excluding integers divisible by 5

```
for (i = 1; i <= n; i++) {
        if (i % 5 == 0) continue;
        sum += 1;
}
```

GOTO

- *goto* allows you to jump *unconditionally* to arbitrary part of your code (within the same function)
- The location is identified using a label
- Using goto makes code harder to read and debug.
- Any code that uses goto can be written without using one
- Language like C++ and Java provide exception mechanism to recover from errors. In C, goto provides a convenient way to exit from nested blocks.

```
for (...) {
        for (...) {
                if (error cond)
                        goto error;
                        /* skip two blocks */
        }
}
error:
```

FUNCTIONS

- Already seen some functions, including *main*():
- Basic syntax:

```
int main(void) {
        /* do stuff */
        return 0; /* success */
}
```

- C only provides *call by value* parameter passing, the called function is only provided with the current values of the arguments.
- *Example 7*
- *Example 8:* Scope Example
- How to modify a value of a variable inside function-body?
  - pointers, pass address
  - use global variables
  - return from the function