# CSE 105, Lab 4, Summer 2006
# cse.unl.edu/∼cstrope/csce105su06/lab/lab4/
# IF IF IF...

Instructor: Cory Strope

July 22, 2006

1. Tools for Decision Making

   - Relational operators $-\ <\ <=\ ==\ >=\ >\ !\ =$
     - Binary operators: Compares 2 values and/or variables
     - Type can be char, double, int.
     - Result should be interpreted as a truth value, 0 for false, non-zero for true
   - logical operators - A&&B !A A||B
     - All but ! are binary operators
     - Work on logical values (0 or nonzero, false or true)
     - Results should be interpreted as a truth value, 0 for false, non-zero for true

2. Control Structure for Decision Making

   - `if` Structure: If the logical expression is **true (nonzero)**, the single executable statement OR block of exectuable statements are executed If logical expression is **false (zero)**, the single executable OR block of executable statement are NOT executed. In other words, the logical statement determines if the code following the statement is skipped or not.

     ```
     if ( logical expression )
        executable statement;
     ```

     OR

```
if ( logical expression )
{
    executable statement;
    executable statement;
}
```

- Compile source code `doll2dougsam1.c` and run entering the number 9 first. Run a second time entering the number 10. Open the source code and study the if statement structure.
- Compile source code `doll2dougsam2.c` and run entering the number 9 first. Run a second time entering the number 10. Open the source code and study the if statement structure. What is the difference between the two programs?.

- `if else` Structure: If the logical expression is **true (nonzero)**, the single executable statement OR block of exectuable statements are executed If logical expression is false (zero), the single executable OR block of executable statement following the else is executed. In other words, the logical statement determine which of two possible directions the program can go in.

```
if ( logical expression )
{
    executable statement;
    executable statement;
}
else
{
    executable statement;
    executable statement;
}
```

- Compile source code `doll2dougsam3.c` and run entering the number 9 first. Run a second time entering the number 10. Open the source code and study the if else statement structure. What is the difference between `doll2dougsam2.c` and `doll2dougsam3.c`?

- `if else if else` Structure: If the first logical expression is **true (nonzero)**, the single executable statement OR block of exectuable statements are executed. If the logical expression is **false (zero)**, the program goes to the next else if logical statement and checks that condition The program will execute only the block where either the logical statement is true or there is no logical statement (else only). In other words, the logical statements determine which of several possible directions the program can go in. If there is no ending else, then no statements are executed as default.

```
if ( logical expression )
{
    executable statement;
    executable statement;
}
else if ( logical expression )
{
    executable statement;
    executable statement;
}
else
{
    executable statement;
    executable statement;
}
```

- Compile source code `doll2dougsam4.c` and run entering the number 9 first. Run a second time entering the number 10. Run a third time entering the number 20. Open the source code and study the if else if else statement structure. What is the difference between `doll2dougsam3.c` and `doll2dougsam4.c`?

- Compile source code `doll2dougsam4_buggy.c` and run entering the number 25 first. Is the result correct. Why or why not! How can it be fixed?

3. Error checking

- We often want to check to see if a user enters a valid input.
- For this program:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int day, month, year;

    printf("Enter the month > ");
    scanf("%d", &month);

    // Check to see if this is a valid month
    if(                                          ) {

        printf("%d is an invalid month!\n");
        exit(0);
    }

    // More code.

    return 0;
}
```

  Write a valid condition to check to see if the input month is valid or not.

- `switch` Statements: We use `switch` statements when there is an integer or character variable that can take only a limited number of values. For example:

```
    switch (controlling_variable)
    {
    case 1:
        executables first group
        break;
    case 2:
        executables second group
```

```
    break;
case 3:
    executable third group
    break;
default:
    executable fourth group
}
```

4. The program `sample.c` is a program that will be used to determine the day number (1–366) in a year for a date that is provided as input data. As an example, January 1, 1994, is day 1. December 31, 1993, is day 365. December 31, 1996, is day 366, since 1996 is a leap year. A year is a leap year if it is divisible by four, except that any year divisible by 100 is a leap year only if it is divisible by 400. `sample.c` accepts the month, day, and year as integers. Change `sample.c` so that it includes a function `leap` that returns 1 if called with a leap year, 0 otherwise. Change the `if-else` statements into an equivalent `switch` statement. Finally, also write a function `valid` that takes the day, month, and year, and returns a 1 if the day/month/year combination is valid, and 0 if it is not. For example, 13/23/2006 is an invalid date, since 13 is not a month; 2/29/1995 is invalid since there are not 29 days in February unless it is a leap year; 4/344/1998 is invalid since there are not 344 days in any month. Not that there is not a case in which the input year is invalid.

## Email programming problems a. and b. by midnight tonight!!