If, else-if, switch-case conditional statements

if ( TRUE ) {
    /* Execute these stmts if TRUE */
}
else {
    /* Execute these stmts if FALSE */
}

if (condition) {
    statement(s); }
else if (condition) {
    statement(s); }
else {
    statement(s); }

switch ( <variable> ) {
    case this-value:     /* Note the ;, not a ; */
        Code to execute if <variable> == this-value;
        break;
    case that-value:
        Code to execute if <variable> == that-value;
        break;
    ... default:
        Code to execute if <variable> does not equal the value following any of the cases break;
}

SWITCH NOTES:
- Notice, no {} blocks within each case.
- Notice the colon for each case and value.
- The “condition” of a switch statement is a value.
- The default case is optional, but it is wise to include it as it handles any unexpected cases.
- Chooses first match...
#include <stdio.h>
int main() {
    int age;
    printf( "Please enter your age" );
    scanf( "%d", &age );
    if ( age < 100 ) {
        printf ("You are pretty young!
" );
    } else if ( age == 100 ) {
        printf( "You are old\n" );
    } else {
        printf( "You are really old\n" );
    }
    return 0;
}

NOTE: You do not have to use {} if only one statement in the block. None of the above brackets in the IF structure are necessary! Check out where the semi-colon goes (and where it doesn’t).
Switch example

```
#include <stdio.h>

void playgame() { printf( "Play game called" ); }
void loadgame() { printf( "Load game called" ); }
void playmultiplayer() { printf( "Play multiplayer game called" ); }

int main() {
    int input;
    printf( "1. Play game\n" );
    printf( "2. Load game\n" );
    printf( "3. Play multiplayer\n" );
    printf( "4. Exit\n" );
    printf( "Selection: " );
    scanf( "%d", &input );
    switch ( input ) {
        case 1:
            playgame();
            break;
        case 2:
            loadgame();
            break;
        case 3:
            playmultiplayer();
            break;
        case 4:
            printf( "Thanks for playing!\n" );
            break;
        default:
            printf( "Bad input, quitting!\n" );
            break;
    }
    getchar();
    return 0;
}
```

```c
switch ( x ) {
    case 'a':
        /* Do stuff when x is 'a' */
        break;
    case 'b':
    case 'c':
    case 'd':
        /* Fallthrough technique...
         * cases b,c,d all use this code */
        break;
    default:
        /* Handle cases when x is not
         * a,b,c or d. ALWAYS have a
         * default case*/
        break; }
```
What is GDB?

GDB: The GNU Project Debugger

Allows you to see what is going on “inside” another program while it executes -- or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act*:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

* or just for fun to see what is going on behind the scenes :o)
Using GDB

- %nl gdbincl.c > gdbinclnl
  - gdbtestnl is a text file so no extension necessary
  - Use an editor to open gdbinclnl
  - Now can reference line numbers
- %more gdbincl.c
  - Shows your program on the screen

COMMANDS
- help – lists gdb command topics
- info xxx – where xxx be to list the breakpoints, breakpoint numbers, registers, etc
- run – starts execution
- quit – short cut is just q
GDB command (cont)

- **Break and watch commands**
  - break/tbreak followed by:
    - Function name, line number
  - clear – delete breakpoints
  - watch – followed by a condition
    -Suspends processing when condition is met
  - delete – delete all break/watch points
  - continue – exec until next break/watch point
  - finish – continue to end of function

- **Line execution commands**
  - step – step to next line of code (will step into a function)
  - next – execute next line of code (will not enter functions)
  - until - Continue processing until you reach a specified line number