Flow Control and Booleans

1. Please get logged in.
2. Open a new terminal window Applications/Accessories/Terminal
3. Open a web browser Applications/Internet/Iceweasel Web Browser
4. Go to http://www.cse.msu.edu/~cse251
5. Open Step 3: Flow Control and Booleans

Today: Flow Control and Booleans

Flow Control

if, switch

Boolean Logic

<, <=, ==, >=, >, !=

rlc.c

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

/*
 * Simple program to compute the resonant frequency of
 * an RLC circuit
 */

int main()
{
    double l;       /* Inductance in millihenrys */
    double c;       /* Capacitance in microfarads */
    double omega;   /* Resonance frequency in radians per second */
    double f;       /* Resonance frequency in Hertz */

    printf("Enter the inductance in millihenrys: ");
    scanf("%lf", &l);

    printf("Enter the capacitance in microfarads: ");
    scanf("%lf", &c);

    omega = 1.0 / sqrt((l / 1000) * (c / 1000000));
    f = omega / (2 * M_PI);
    printf("Resonant frequency: %.2f\n", f);
}
```

rlc.c

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

/*
 * Simple program to compute the resonant frequency of
 * an RLC circuit
 */

int main()
{
    double l;       /* Inductance in millihenrys */
    double c;       /* Capacitance in microfarads */
    double omega;   /* Resonance frequency in radians per second */
    double f;       /* Resonance frequency in Hertz */

    printf("Enter the inductance in millihenrys: ");
    scanf("%lf", &l);

    printf("Enter the capacitance in microfarads: ");
    scanf("%lf", &c);

    omega = 1.0 / sqrt((l / 1000) * (c / 1000000));
    f = omega / (2 * M_PI);        /* Convert radians per sec to Hertz */
    printf("Resonant frequency: %.2f\n", f);
}
```

Note the use of comments to tell: a) what the program does, b) what some lines of code do, and c) what the variables are.

rlc.c

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

/*
 * Simple program to compute the resonant frequency of
 * an RLC circuit
 */

int main()
{
    double l;       /* Inductance in millihenrys */
    double c;       /* Capacitance in microfarads */
    double omega;   /* Resonance frequency in radians per second */
    double f;       /* Resonance frequency in Hertz */

    printf("Enter the inductance in millihenrys: ");
    scanf("%lf", &l);

    printf("Enter the capacitance in microfarads: ");
    scanf("%lf", &c);

    omega = 1.0 / sqrt((l / 1000) * (c / 1000000));
    f = omega / (2 * M_PI);
    printf("Resonant frequency: %.2f\n", f);
}
```

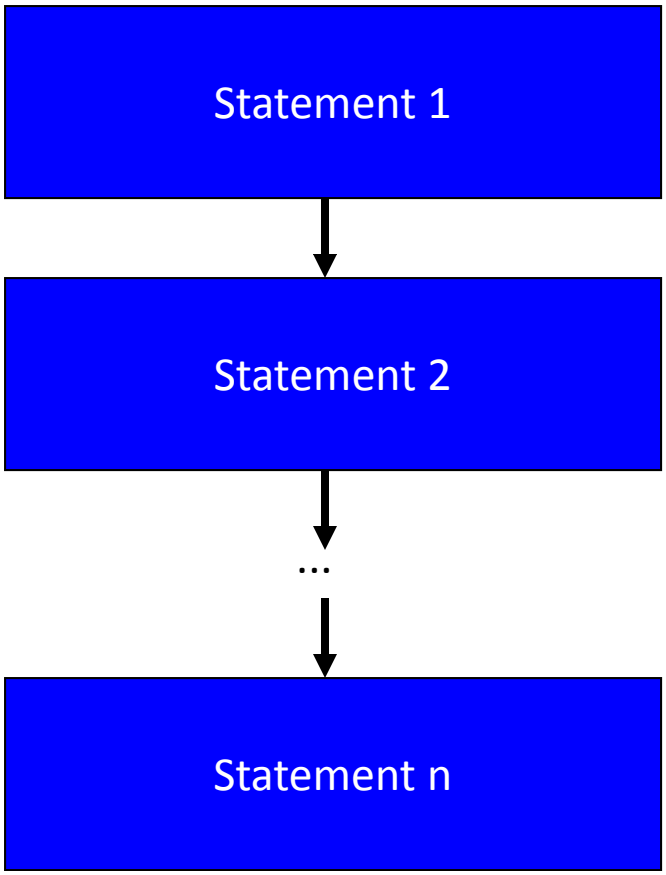
Note the use of indentation

rlc.c

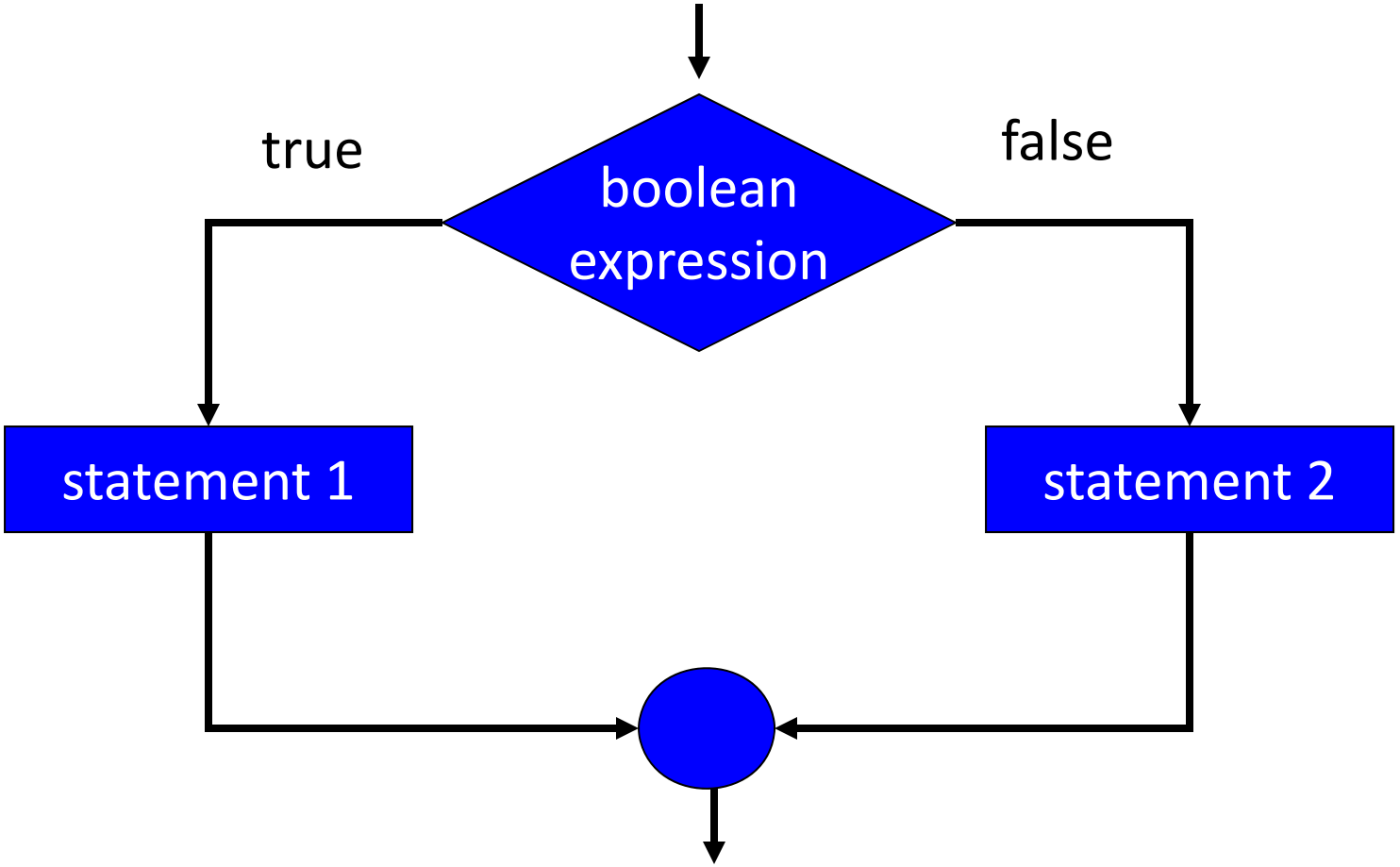
```
#include <stdio.h>
#include <math.h>

/*
 * Simple program to compute the resonant frequency of
 * an RLC circuit
 */

int main()
{
double l; /* Inductance in millihenrys */
double c; /* Capacitance in microfarads */
double omega; /* Resonance frequency in radians per second */
double f; /* Resonance frequency in Hertz */
printf("Enter the inductance in millihenrys: ");
scanf("%lf", &l);
printf("Enter the capacitance in microfarads: ");
scanf("%lf", &c);
omega = 1.0 / sqrt((l / 1000) * (c / 1000000));
f = omega / (2 * M_PI);
printf("Resonant frequency: %.2f\n", f);
}
```

Indentation is necessary to make your program readable!

Sequential Execution



Selective Execution



if statements

```
if(age > 39)
   printf("You are so old!\n");
```

The if statement

Fundamental means of *flow control*

How we will make decisions

Boolean expressions

The actual determination of the decision

```
age > 39
c == 0
l <= 0
(age >= 18) && (age < 65)
```

Structure of an if statement

```
if(expression1)
   statement1;
else if(expression2)          /* Optional */
   statement2;
else                          /* Optional */
   statement3;
```

If expression1 is true, execute statement1.

Otherwise, test to see if expression2 is true. If so, execute statement2.

Otherwise, execute statement3.

The expressions are *boolean expressions* that resolve to a true or a false.

Basic Boolean Expressions

true
false
age < 18
divisor == 0
size > 1000000
ch == 'X'

Some operators:

<	Less than
<=	Less than or equal to
==	Equal
!=	Not equal
>=	Greater than or equal to
>	Greater than

Important: The test for equality is ==, not =. This is the most common error in a C program.

Example if statements

<	Less than
<=	Less than or equal to
==	Equal
!=	Not equal
>=	Greater than or equal to
>	Greater than

```
if(age < 18)
   printf("Too young to vote!\n");
```

```
if(area == 0)
   printf("The plot is empty\n");
else
   printf("The plot has an area of %.1f\n", area);
```

```
if(val < 0)
   printf("Negative input is not allowed\n");
else if(val == 0)
   printf("A value of zero is not allowed\n");
else
   printf("The reciprocal is %.2f\n", 1.0 / val);
```

Note the indentation

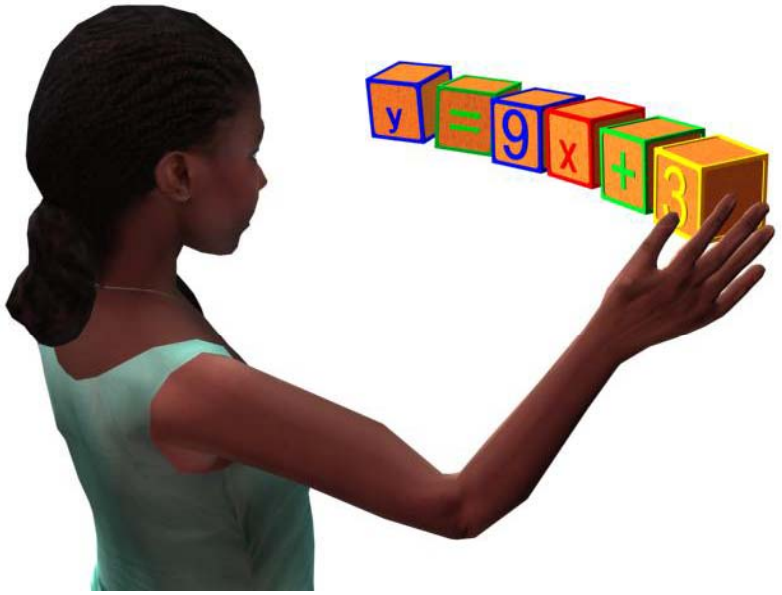
Blocks

Single Statement

```
printf("This is a statement\n");
```

Block

```
{
    printf("All items in a curly brace\n");
    printf("as if there are one statement");
    printf("They are executed sequentially");
}
```



Where is this useful?

If the expression is true, all of the statements in the block are executed

```
if(value > 0)
{
    result = 1.0 / value;
    printf("Result = %f\n", result);
}
```

Where is this useful?

Will these two sections of code work differently?

```
if(value > 0)
{
    result = 1.0 / value;
    printf("Result = %f\n", result);
}
```

```
if(value > 0)
    result = 1.0 / value;
    printf("Result = %f\n", result);
```

Where is this useful?

Yes!

```
if(value > 0)
{
    result = 1.0 / value;
    printf("Result = %f\n", result);
}
```

Will always execute!

```
if(value > 0)
    result = 1.0 / value;
    printf("Result = %f\n", result);
```

Nested Blocks

What does this do?

```
if(bobsAge != suesAge)  /* != means "not equal" */
{
    printf("Bob and Sue are different ages\n");
    if(bobsAge > suesAge)
    {
        printf("In fact, Bob is older than Sue\n");
        if((bobsAge - 20) > suesAge)
        {
            printf("Wow, Bob is more than 20 years older\n");
        }
    }
}
```

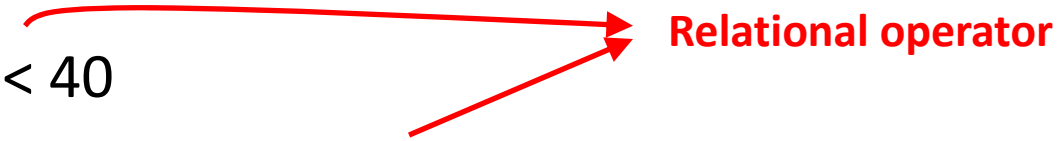
Importance of indentation

See how much harder this is to read?

```
if(bobsAge != suesAge)  /* != means "not equal" */
{
printf("Bob and Sue are different ages\n");
if(bobsAge > suesAge)
{
printf("In fact, Bob is older than Sue\n");
if((bobsAge - 20) > suesAge)
{
printf("Wow, Bob is more than 20 years older\n");
}
}
}
```

Boolean Expressions

- An expression whose value is true or false
- In C:
  - integer value of 0 is “false”
  - nonzero integer value is “true”
- Example of Boolean expressions:
  - age < 40
  - graduation_year == 2010



```c
#include <stdio.h>
#include <stdbool.h>

int main()
{
    const bool trueVar = true, falseVar = false;
    const int int3 = 3, int8 = 8;

    printf("No 'boolean' output type\n");
    printf("bool trueVar: %d\n",trueVar);
    printf("bool falseVar: %d\n\n",falseVar);
    printf("int int3: %d\n",int3);
    printf("int int8: %d\n",int8);
}
```

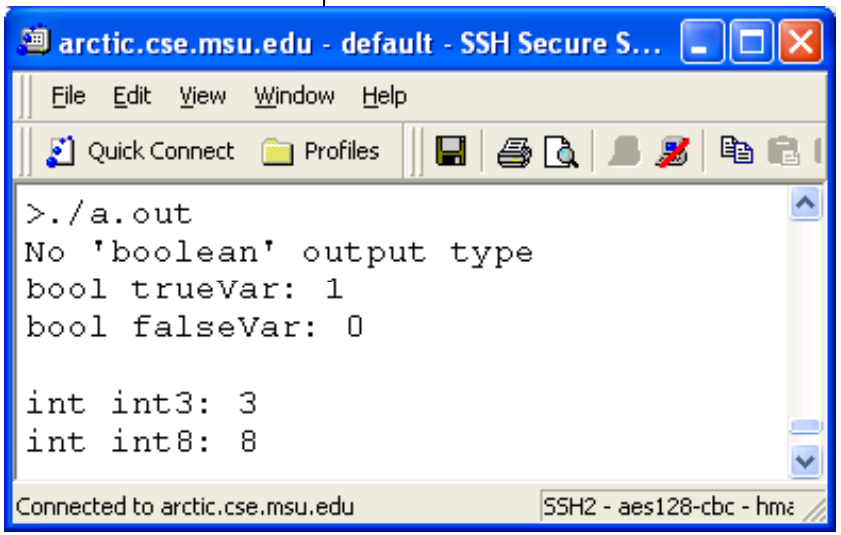
Library that defines: bool, true, false

What does the output look like?

```c
#include <stdio.h>
#include <stdbool.h>

int main()
{
    const bool trueVar = true, falseVar = false;
    const int int3 = 3, int8 = 8;

    printf("No 'boolean' output type\n");
    printf("bool trueVar: %d\n",trueVar);
    printf("bool falseVar: %d\n\n",falseVar);
    printf("int int3: %d\n",int3);
    printf("int int8: %d\n",int8);
}
```

Library that defines: bool, true, false

What does the output look like?



// Example3 (continued…)

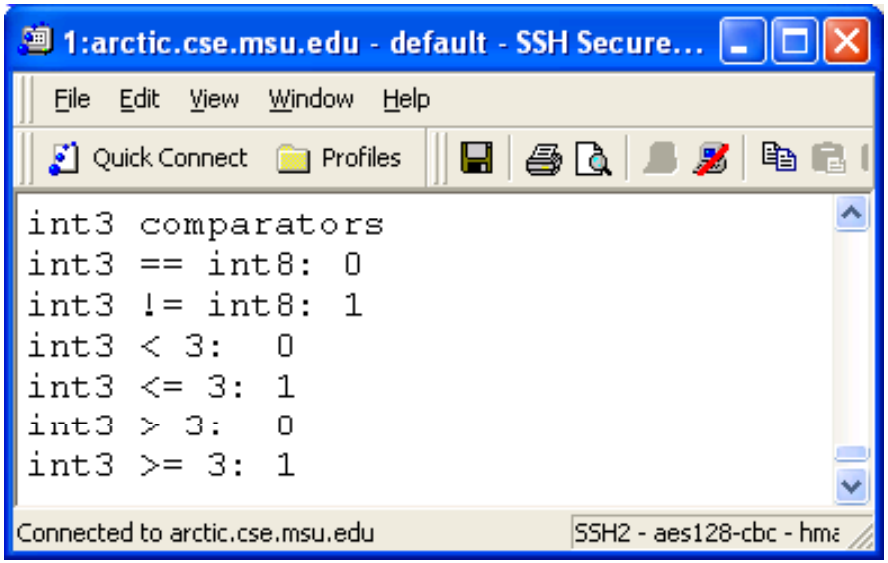
```
printf("\nint3 comparators\n");
printf("int3 == int8: %d\n",(int3 == int8));
printf("int3 != int8: %d\n",(int3!=int8));
printf("int3 < 3:  %d\n",(int3 <  3));
printf("int3 <= 3: %d\n",(int3 <= 3));
printf("int3 > 3:  %d\n",(int3 >  3));
printf("int3 >= 3: %d\n",(int3 >= 3));
```

Comparing values of two integer constants

What does the output look like?

// Example3 (continued…)

```
printf("\nint3 comparators\n");
printf("int3 == int8: %d\n",(int3 == int8));
printf("int3 != int8: %d\n",(int3!=int8));
printf("int3 < 3:  %d\n",(int3 <  3));
printf("int3 <= 3: %d\n",(int3 <= 3));
printf("int3 > 3:  %d\n",(int3 >  3));
printf("int3 >= 3: %d\n",(int3 >= 3));
```



More Examples

- char myChar = ‘A’;
  - The value of myChar==‘Q’ is false (0)

- Be careful when using floating point equality comparisons, especially with zero, e.g. myFloat==0

Suppose?

What if I want to know if a value is in a range?

Test for: $100 \leq L \leq 1000$?

You can't do…

This code is WRONG and will fail.

```
if(100 <= L <= 1000)
{
    printf("Value is in range…\n");
}
```

Why this fails…


```
if((100 <= L) <= 1000)
{
    printf("Value is in range…\n");
}
```

Suppose L is 5000. Then 100 <= L is true, so (100 <= L) evaluates to true, which, in C, is a 1. Then it tests 1 <= 1000, which also returns true, even though you expected a false.

Compound Expressions

- Want to check whether -3 <= B <= -1
  - Since B = -2, answer should be True (1)
- But in C, the expression is evaluated as
  - ((-3 <= B) <= -1) (<= is left associative)
  - (-3 <= B) is true (1)
  - (1 <= -1) is false (0)
  - Therefore, answer is 0!

Compound Expressions

- Solution (not in C): (-3<=B) and (B<=-1)

- In C: (-3<=B) && (B<=-1)
- Logical Operators
  - And: &&
  - Or: ||
  - Not: !

Compound Expressions

```
#include <stdio.h>

int main()
{
   const int A=2, B = -2;

   printf("Value of A is %d\n", A);
   printf("0 <= A <= 5?: Answer=%d\n", (0<=A) && (A<=5));

   printf("Value of B is %d\n", B);
   printf("-3 <= B <= -1?: Answer=%d\n", (-3<=B) && (B<=-1));
}
```

Compound Expressions

```
#include <stdio.h>

int main()
{
   const int A=2, B = -2;

   printf("Value of A is %d\n", A);
   printf("0 <= A <= 5?: Answer=%d\n", (0<=A) && (A<=5));

   printf("Value of B is %d\n", B);
   printf("-3 <= B <= -1?: Answer=%d\n", (-3<=B) && (B<=-1));

}
```

True (1) True (1)

```
>./a.out
Value of A is 2
0 <= A <= 5?: Answer=1
Value of B is -2
-3 <= B <= -1?: Answer=1
```

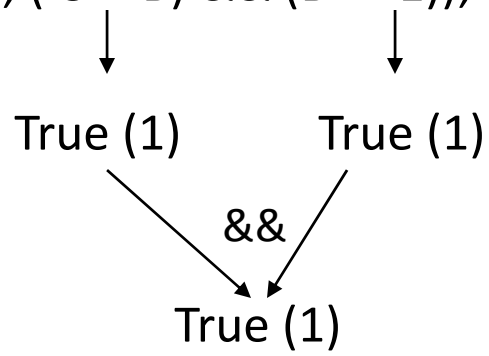
Correct Answer!!!

Compound Expressions

```
#include <stdio.h>

int main()
{
   const int A=2, B = -2;

   printf("Value of A is %d\n", A);
   printf("0 <= A <= 5?: Answer=%d\n", (0<=A) && (A<=5));

   printf("Value of B is %d\n", B);
   printf("-3 <= B <= -1?: Answer=%d\n", (-3<=B) && (B<=-1));

}
```

```
>./a.out
Value of A is 2
0 <= A <= 5?: Answer=1
Value of B is -2
-3 <= B <= -1?: Answer=1
```

Correct Answer!!!



Truth Tables

p	q	Not !p	And p && q	Or p || q
True	True			
True	False			
False	True			
False	False			

Truth Tables

p	q	Not !p	And p && q	Or p || q
True	True	False		
True	False	False		
False	True	True		
False	False	True		

Truth Tables

p	q	Not !p	And p && q	Or p || q
True	True		True	
True	False		False	
False	True		False	
False	False		False	

Truth Tables

p	q	Not !p	And p && q	Or p || q
True	True			True
True	False			True
False	True			True
False	False			False

Truth Tables

Our comparison operators:
< <= == != >= >

p	q	!p (Not)	p && q (And)	p || q (Or)
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	False

Precedence & Associativity

```
>cat compound3.c
#include <stdio.h>

int main() {
    int A = 4, B = 2;

    printf("Answer is %d\n", A + B > 5 && (A = 0) < 1 > A + B - 2);
}
```

Can you guess what's the answer?

Relational operators have precedence and associativity (just like arithmetic operators)

Use ( ) when in doubt

```
A = 4, B = 2;
A + B > 5 && (A = 0) < 1 > A + B - 2
```

(((A + B) > 5) && (((A=0) < 1) > ((A + B) – 2)))

((6 > 5) && (((A=0) < 1) > ((A + B) – 2)))

(1 && ((0 < 1) > ((A + B) – 2)))

(1 && (1 > (2 – 2)))

(1 && (1 > 0))

(1 && 1)

Answer: 1

Precedence: +/- > < &&

Associativity

"=" is right associative

Example: X=Y=5

right associative: X = (Y=5)

expression Y=5 returns

value 5: X = 5

You should refer to the C operator precedence and associative table

See for example, http://www.difranco.net/cop2220/op-prec.htm

Or just use parentheses whenever you're unsure about precedence and associativity

Operator	Description	Associativity

`()`
`[]`
`.`
`->`
`++ --`
	Parentheses (function call) (see Note 1)
	Brackets (array subscript)
	Member selection via object name
	Member selection via pointer
	Postfix increment/decrement (see Note 2)
		left-to-right

`++ --`
`+ -`
`! ~`
`(type)`
`*`
`&`
`sizeof`
	Prefix increment/decrement
	Unary plus/minus
	Logical negation/bitwise complement
	Cast (change *type*)
	Dereference
	Address
	Determine size in bytes
		right-to-left

`* / %`	Multiplication/division/modulus	left-to-right
`+ -`	Addition/subtraction	left-to-right
`<< >>`	Bitwise shift left, Bitwise shift right	left-to-right
`< <=` `> >=`	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
`== !=`	Relational is equal to/is not equal to	left-to-right
`&`	Bitwise AND	left-to-right
`^`	Bitwise exclusive OR	left-to-right
`|`	Bitwise inclusive OR	left-to-right
`&&`	Logical AND	left-to-right
`||`	Logical OR	left-to-right
`?:`	Ternary conditional	right-to-left

`=`
`+= -=`
`*= /=`
`%= &=`
`^= |=`
`<<= >>=`
	Assignment
	Addition/subtraction assignment
	Multiplication/division assignment
	Modulus/bitwise AND assignment
	Bitwise exclusive/inclusive OR assignment
	Bitwise shift left/right assignment
		right-to-left

`,`	Comma (separate expressions)	left-to-right

Switch Statement

A less general substitute for the multibranch if. It is used for selecting among discrete values (int), i.e. not continuous values.

```
switch (int_expression)
{
    case_list:
          statement_list;
    case_list:
          statement_list;
    default:
          statement_list;
}
```

Behavior

- The `int_expression` is evaluated. If the value is in a `case_list`, execution begins at that `statement_list` and continues through subsequent `statement_lists` until: `break`, `return`, or end of switch.

```c
#include <stdio.h>

void main()
{
   int gender;
   printf("Enter your gender (male=1, female=2): ");
   scanf("%d",&gender);

   switch(gender)
   {
        case 1:
           printf("You are a male\n");
           break;
        case 2:
           printf("you are a female\n");
           break;
        default:
           printf("Not a valid input\n");
           break;
   }
}
```

```
switch(gender)
{
    case 1:
       printf("You are a male\n");
       break;
    case 2:
       printf("you are a female\n");
       break;
    default:
       printf("Not a valid input\n");
       break;
}
```