CSE / ENGR 142 Programming I

Conditionals

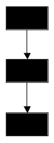
Chapter 4

Read Sections 4.1-4.5, 4.7-4.9

- 4.1: Control structure preview
- 4.2: Relational and logical operators
- 4.3: `if` statements
- 4.4: Compound statements
- 4.5: Example
- 4.7: Nested `if` statements
- 4.8: `switch` statements

Preview of Things to Come

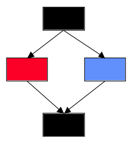
- "Control flow" is the order in which statements are executed
- Until now, control flow has been sequential -- the next statement executed is the next one that appears, in order, in the C program



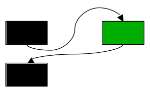
Preview Prologue

We're going to look at two ways to indicate non-sequential control flow

"conditionals," which pick one of two (or sometimes more) next statements



"procedures" / "subroutines" / "functions", which allows you to "visit" a chunk of code and then come back



Conditional Execution

- A conditional statement allows the computer to <u>choose</u> an execution path depending on the value of a variable or expression
  - if the withdrawal is more than the bank balance, then print an error
  - if today is my birthday, then add one to my age
  - if my grade is greater than 3.5, then attend party

Conditional ("*if* ") Statement

```
if (condition) statement;
```

The statement is executed if and only if the condition is true.

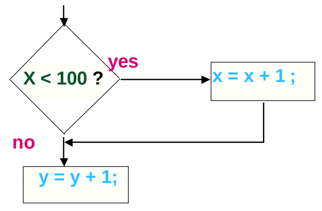
if (x < 100) x = x + 1;

if (withdrawalAmount > balance) printf("NSF check.\n");

if (temperature > 98.6) printf("You have a fever.\n");

Conditional Flow Chart

```
if (x < 100) x = x + 1 ;
y = y + 1;
```



Conditional Expressions

- Also called "logical" or "Boolean" expressions
- Made up of variables, constants, arithmetic expressions, and the "relational operators":

Math symbols:	< , ≤ , > , ≥ , = , ≠
in C:	< , <=, > , >= , == , !=

air_temperature > 0.0
98.6 <= body_temperature
marital_status == 'M'
divisor != 0

some conditional expressions

Such expressions are used in "if" statements and numerous other places in C.

Value of conditional expressions

- Remember that "expressions are things that have a value."
- What is the value of a conditional expression??
- Answer: we think of it as TRUE or FALSE
  - Most of the time, TRUE or FALSE is all you have to think about - and how you should think about it.
- Under the hood in C, it's really an integer
  - FALSE is 0 *(and 0 is FALSE)*
  - TRUE is any value other than 0 *(and non-zero is TRUE)*
    - frequently 1
    - 1 is result of relational operator (<, <=, >=, ==, !=) when relation is true

Complex Conditionals

- if I have at least $15 or you have at least $15, then we can go to the movies
- if the temperature is below 32 degrees and it's raining, then it's snowing
- if it's not the case that it's Saturday or Sunday, then it's a work day

Complex Conditionals in C

Boolean operators	&&	||	!
	and	or	not

```
#define TRUE   1
#define FALSE  0

if (myMoney>=15.0 || yourMoney>=15.0) canGoToMovies = TRUE;

if (temperature<32.0 && raining) snowing = TRUE;

weekday = TRUE;
if (!(today==6 || today==7)) weekday = FALSE;
if (weekday) mustWork = TRUE;
```

More about this topic later!

Multiple actions

More than one conditional action?
Use a compound statement:

```
if ( temperature > 98.6 ) {
  printf ( "You have a fever. \n" );
  aspirin  =  aspirin – 2 ;
}
```

Compound Statement

- Also called a "block."
- Groups together statements so that they are treated as a single statement:

```
{
    statement1 ;
    statement2 ;
    ...
}
```

- Highly useful
  - Not just in conditionals, but many places in C

Principles for combining and substituting statements

1. *You may use a compound statement* ***anywhere*** *that a single statement may be used.*

2. *Anywhere that a statement is allowed in C,* ***any kind*** *of statement can be used.*

3. *A compound statement man contain* ***any number*** *of statements (including 0)*

Among other things, these principles imply that compound statements can be nested to any depth.

Compound Example

Cash machine program fragment:

```
if ( balance >= withdrawal ) {
   balance = balance - withdrawal ;
   dispense_funds ( withdrawal ) ;
}
```

- Puzzlers:
  - What if { } omitted?
  - What if ( ) omitted?

Finding Absolute Value

Problem: Compute the absolute value |x| of x and put the answer in variable *abs*. Here are three solutions, all correct:

```
if (x >= 0) abs = x;
if ( x < 0 ) abs =  -x;
```

```
abs = x;
if ( x < 0 ) abs  = -x;
```

```
if (x >= 0) abs = x;
else abs = -x;
```

Absolute Value as a Function

P.S.: A better approach is to define a function to compute absolute value |x|:

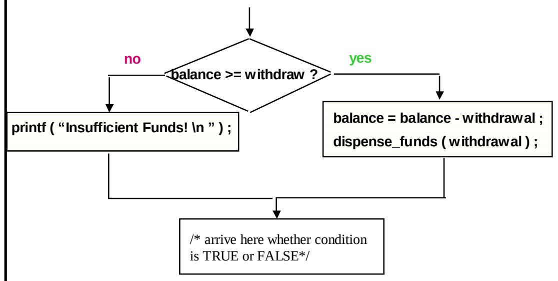
```
int abs ( int x )
{
   if ( x < 0 )
      x  =  – x ;
   return ( x ) ;
}
```

An expanded type of conditional: *if - else*

Print error message:

```
if ( balance >= withdrawal ) {
   balance = balance - withdrawal ;
   dispense_funds ( withdrawal ) ;
}       no ; here
else {
   printf ( “Insufficient Funds! \n ” ) ;
}
```

if - else Control Flow


no
yes
balance >= withdraw ?
printf (“Insufficient Funds! \n ”) ;
balance = balance - withdrawal ;
dispense_funds (withdrawal) ;
/* arrive here whether condition is TRUE or FALSE*/

Nested *ifs*

```
#define BILL_SIZE  20
if ( balance >= withdrawal ) {
    balance = balance - withdrawal ;
    dispense_funds ( withdrawal ) ;
} else {
    if ( balance >= BILL_SIZE ) printf ( “Try a smaller amount. \n ” ) ;
    else printf ( “Go away! \n ” ) ;
}
```

Nested *ifs* , Part II

```
if ( x == 5 ) {
    if  ( y == 5 ) printf ( “Both are 5. \n ”) ;
    else printf ( “x is 5, but y is not. \n ”) ;
} else {
    if  ( y == 5 ) printf ( “y is 5, but x is not. \n ”) ;
    else printf ( “Neither is 5. \n ”) ;
}
```

Tax Example (Study at Home)

Print the % tax based on income:

income	tax
< 15,000	0%
15,000, < 30,000	18%
30,000, < 50,000	22%
50,000, < 100,000	28%
100,000	31%

Simple Solution

```
if ( income < 15000 ) {
    printf( “No tax.” );
}
if ( income >= 15000 && income < 30000 ) {
    printf(“18%% tax.”);
}
if ( income >= 30000 && income < 50000 ) {
    printf(“22%% tax.”);
}
if ( income >= 50000 && income < 100000 ) {
    printf(“28%% tax.”);
}
if ( income >=100000) {
    printf(“31%% tax.”);
}
```

Mutually exclusive conditions - only one will be true

Cascaded ifs

```
if ( income < 15000 ) {
    printf( “No tax” );
} else {
    if ( income < 30000 ) {
        printf( “18%% tax.” );
    } else {
        if ( income < 50000 ) {
            printf( “ 22%% tax.” );
        } else {
            if ( income < 100000 ) {
                printf( “28%% tax.” );
            } else {
                printf( “31%% tax.” );
            }
        }
    }
}
```

```
if ( income < 15000 ) {
    printf( “No tax” );
} else if ( income < 30000 ) {
    printf( “18%% tax.” );
} else if ( income < 50000 ) {
    printf( “ 22%% tax.” );
} else if ( income < 100000 ) {
    printf( “28%% tax.” );
} else
    printf( “31%% tax.” );
}
```

Order is important. Conditions are evaluated in order given.

Problem: The First Character

```
/* Problem: read 3 characters; print the smallest */
char c1, c2, c3, first;
printf ( "Enter 3 chars> " ) ;
scanf ( "%c%c%c", &c1, &c2, &c3 ) ;
first = c1 ;
if ( c2 < first )
      first = c2 ;
if ( c3 < first )
      first = c3 ;
printf ( "Alphabetically, the first of the 3 is %c",
         first ) ;
```

```
c1  c2  c3  first
?   ?   ?   ?

'h' 'a' 't'  ?
'h' 'a' 't' 'h'
   ( true )
'h' 'a' 't' 'a'
   ( false )
     ---

 ( prints 'a')
```

Function *FirstCharacter*

```
char FirstCharacter(char c1, char c2, char c3)
{
   char first ;
   first = c1 ;
   if ( c2 < first )
        first = c2 ;
   if ( c3 < first )
        first = c3 ;
   return(first);
}
```

Problem: Sort 2 Characters

Top Level View:

Input two characters

Rearrange them in sorted order

Output them in sorted order

Examples:

```
Input:      ra     Output:ar
Input:      nt     Output:nt
```

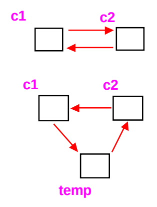
Sort 2 Characters: Algorithm Refinement

```
Input c1, c2
If c2 comes before c1 in alphabet
     Swap c1 and c2
Output c1, c2
```

Why not
c1 = c2;
c2 = c1;
?

Swap

```
Input c1, c2
If c2 comes before c1 in alphabet
     Save c1 in temporary
     Assign c2 to c1
     Assign temporary to c2
Output c1, c2
```



Sort 2 Characters Code

```
/* sort 2 characters and print in sorted order */
char c1, c2, temp ;
printf ( "Enter  2 chars: " ) ;
scanf ( "%c%c", &c1, &c2 ) ;
if ( c2 < c1 ) {    /* swap if out of order */
  temp = c1 ;
  c1 = c2 ;
  c2 = temp ;
}
printf ( "In alphabetical order, they are %c%c",
     c1, c2 ) ;
```

c1	c2	temp
?	?	?
'd'	'a'	?
'd'	'a'	'd'
'a'	'a'	'd'
'a'	'd'	'd'

(prints "ad")

Discussion Question

- What if we wanted to make a function *SortTwoChars*?
  - It would be a nice abstraction
  - We did this kind of abstraction with *abs* and *FirstCharacter*
- Any problem??

Complex Conditions

- AND (&&), OR (||), NOT (!) can be used to make more complicated conditions
- Review: like arithmetic expressions, conditional expressions have a value:
  - TRUE (non-zero) or FALSE (zero)
  - When using relational (<, ==, etc.) and Boolean (&&, ||, !) operators: TRUE is 1; FALSE is 0
  - values are actually *int* (C has no Boolean type). Can be used in int expressions:

```
m = (z >= 0.0) ;     /* means "m gets 1 if z  is positive" */
```

Nested *if* vs. AND (&&)

```
if ( age < 25 ) {
    if ( sex == 'M' ) {
        insurance_rate  =  insurance_rate * 2 ;
    }
}
```

```
if ( (age < 25)   &&   (sex == 'M') ) {
    insurance_rate  =  insurance_rate * 2 ;
}
```

AND (&&), OR (||)

```
if ( (dwi  >  0)  ||  (tickets > 3) ) {
    insurance_rate  =  insurance_rate * 2 ;
}
```

```
/*An int variable can hold a conditional value: */
/* We call such a variable a flag. */

int  high_risk ;
...
high_risk =  ( age < 25  &&  sex  ==  'M' ) ;
if ( high_risk ) insurance_rate  =  insurance_rate * 2 ;
```

Truth Tables for &&, ||

A "truth table" lists all possible combinations of values, and the result of each combination

P	Q	P && Q	P || Q
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

P and Q stand for any conditional expressions

Truth Table for NOT (!)

```
int high_risk ;
...
high_risk =  (age < 25  &&  sex  ==  'M' ) ;
if ( high_risk ) {
} else {
    printf ( "Cheap rates. \n") ;
}
```

```
if ( ! high_risk ) {
    printf ( "Cheap rates. \n") ;
}
```

P	!P
T	F
F	T

DeMorgan's Laws

```
if ( ! (age < 25  &&  sex  ==  'M' ) ) printf ( "Cheap rates. \n") ;
```

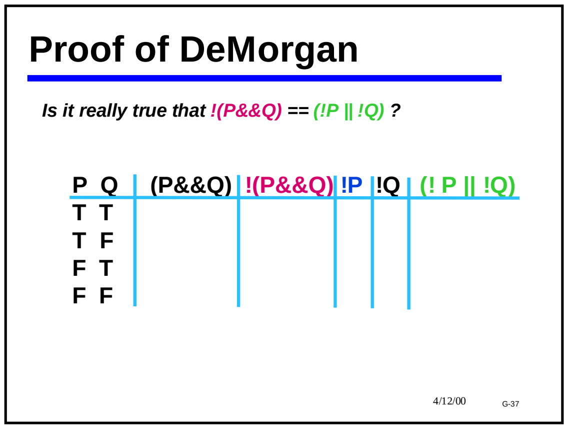
is equivalent to

```
if ( age >= 25  ||  sex  !=  'M' ) ) printf ( "Cheap rates. \n") ;
```

More generally, DeMorgan's laws help determine when two complex conditions are equivalent:

! (P && Q) is equivalent to (!P || !Q)

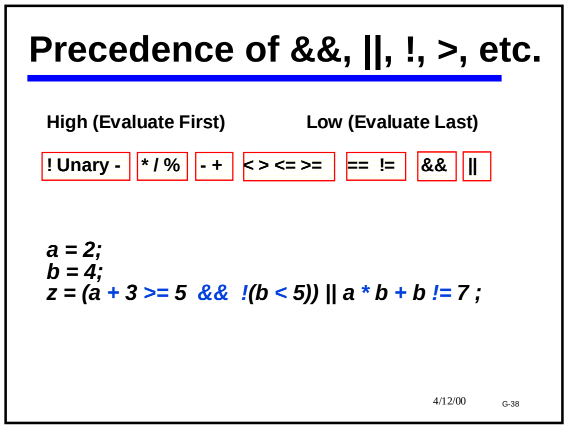
! (P || Q) is equivalent to (!P && !Q)



Proof of DeMorgan

Is it really true that !(P&&Q) == (!P || !Q) ?

P	Q	(P&&Q)	!(P&&Q)	!P	!Q	(! P || !Q)
T	T					
T	F					
F	T					
F	F					

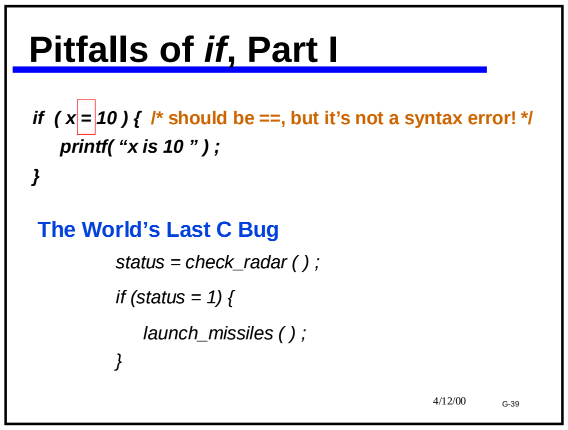


Precedence of &&, ||, !, >, etc.

High (Evaluate First) **Low (Evaluate Last)**

! Unary -	* / %	- +	< > <= >=	== !=	&&	||

```
a = 2;
b = 4;
z = (a + 3 >= 5  &&  !(b < 5)) || a * b + b != 7 ;
```

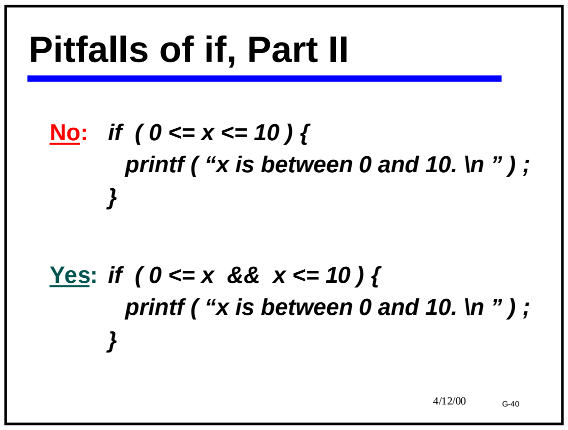


Pitfalls of *if*, Part I

```
if  ( x = 10 ) {  /* should be ==, but it's not a syntax error! */
    printf( "x is 10 " ) ;
}
```

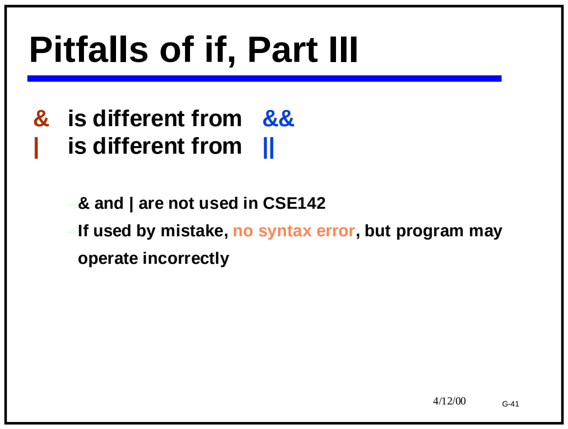
The World's Last C Bug

```
status = check_radar ( ) ;
if (status = 1) {
    launch_missiles ( ) ;
}
```



Pitfalls of if, Part II

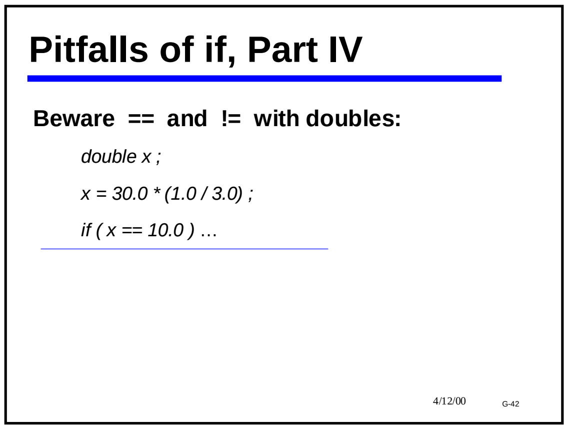
```
No:  if  ( 0 <= x <= 10 ) {
        printf ( "x is between 0 and 10. \n " ) ;
     }

Yes: if  ( 0 <= x  &&  x <= 10 ) {
        printf ( "x is between 0 and 10. \n " ) ;
     }
```



Pitfalls of if, Part III

& is different from &&
| is different from ||

& and | are not used in CSE142
If used by mistake, no syntax error, but program may operate incorrectly



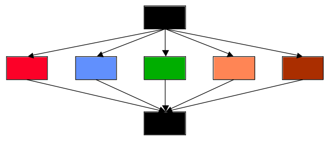
Pitfalls of if, Part IV

Beware == and != with doubles:

```
double x ;
x = 30.0 * (1.0 / 3.0) ;
if ( x == 10.0 ) …
```

Another Control Flow Statement

We're about to switch gears to talk about another kind of control flow statement, the *switch* statement



Longwinded *if*

```
/* How many days in a month? */

if  ( month ==  1 ) {                /* Jan */
     days = 31 ;
} else if ( month ==  2 ) {          /* Feb */
     days = 28 ;
} else if ( month ==  3 ) {          /* Mar */
     days = 31 ;
} else if ( month ==  4 )            /* Apr */
     days = 30 ;
...                                  /* need 12 of these */
```

Clearer Style

```
if  ( month == 9 || month == 4 ||       /* Sep, Apr */
      month == 6 || month == 11 ) {     /* Jun, Nov */
     days = 30 ;
} else if  ( month == 2 ) {             /* Feb */
     days = 28 ;
} else {
     days = 31;                         /* All the rest  */
}
```

Clearest: *switch*

```
/* How many days in a month? */

switch ( month ) {
case 2:                       /* February         */
     days = 28 ;
     break ;
case 9:                       /* September        */
case 4:                       /* April            */
case 6:                       /* June             */
case 11:                      /* November         */
     days = 30 ;
     break ;
default:                      /* All the rest have 31 ...   */
     days = 31 ;
}
printf ( "There are %d days in that month. \n ", days ) ;
```

switch: Flow of Control

```
month = 6 ;
switch ( month )  {
case 2:                           /* February          */
     days = 28 ;
     break ;
case 9:                           /* September         */
case 4:                           /* April             */
case 6:                           /* June              */
case 11:                          /* November          */
     days = 30 ;
     break ;
default:                          /* All the rest have 31 ... */
     days = 31 ;
}
printf ( "There are %d days in that month. \n ", days ) ;
```

switch

```
switch (control expression)
{
case-list1
     statements1
     break;
case-list2
     statements2
     break;
.
.
default:
     statements
}
```

a "case-list" is a series of one or more "case"s

```
case constant1:
case constant2:
.
.
case constantN:
```

The One Big Pitfall of *switch*

```
month = 6 ;
switch (month) {
case 2:                              /* February              */
    days = 28 ;             /* break missing */
case 9:                              /* September             */
case 4:                              /* April                 */
case 6:                              /* June                  */
case 11:                             /* November              */
    days = 30 ;             /* break missing */
default:                             /* All the rest have 31 ...*/
    days = 31 ;
}
printf ( "There are %d days in that month. \n ", days ) ;
```

4/12/00 G-49

switch on char is Legal!

```
char marital_status ;
...
switch ( marital_status ) {
case 'm':
case 'M':
    printf ( "Married \n" ) ;
    break ;                              int or char expression
case 's':
case 'S':
    printf ( "Single \n" ) ;
    break ;
default:
    printf ( "Sorry, I don't recognize that code. \n" ) ;
}
```

4/12/00 G-50

Conditionals: Summary

no ";"

- if (logical expression) {
  the "then" statements
  }
- if (logical expression) {
  the "then" statements
  } else {
  the "else" statements
  }

Parens

- comparisons < <= > >= == !=
- combining && || !
- DeMorgan's Laws
- switch: several cases based on single int or char value

4/12/00 G-51