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

Conditional Execution

- The power of computers lies to a large extent with conditionals.
- A conditional statement allows the computer to choose a different execution path depending on the value of a variable or expression, e.g.:
  - if the withdrawal is more than the bank balance, print an error
  - if today is my birthday, add one to my age
  - if my grade is greater than 3.5, go to party
  - if x is bigger than y then store x in z otherwise store y in z

Conditional ("*if* ") Statement

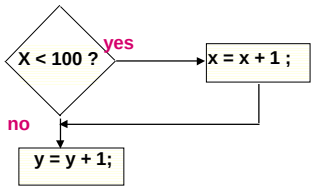
`if (condition) statement ;`

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

```
if (x < 100) x = x + 1 ;
if (withdrawal_amt > balance) print_error( ) ;
if (temperature > 98.6)
        printf("You have a fever.\n");
if ( month == birthmonth && day == birthday)
        my_age = my_age + 1 ;
```


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":

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

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

some conditional expressions

Boolean Operators in Conditional Expressions

Boolean operators && (and) || (or) ! (not)

Examples:

temp > 90.0 && humidity > 50.0

! (salary < 30000 || exemptions < 4)

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
  - TRUE is any value other than 0
    - frequently 1
    - 1 is result of relational operator (<, <=, >=, ==, !=) when relation is true

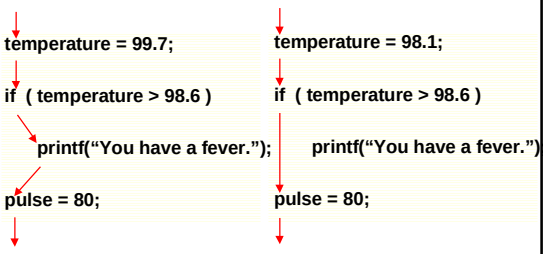
if statements

The semantics of *if* in C follow the English meaning pretty well.

Conditionally execute a statement if the conditional expression evaluates to TRUE (non-zero)

```
if ( temperature > 98.6 || blood_pressure > 160 )
    printf( "You are sick.\n" );
```


Flow of Control



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

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

- Can contain any number of statements (including none)
  - { } is a valid statement!
- Can include any kind of statements.

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.*

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 ) ;
 }
```

- **What if { } omitted?**
- **What if ( ) omitted?**

Another Example

Compute the absolute value |x| of x and put the answer in variable *abs*:

```
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

Function to Compute Absolute Value |x|:

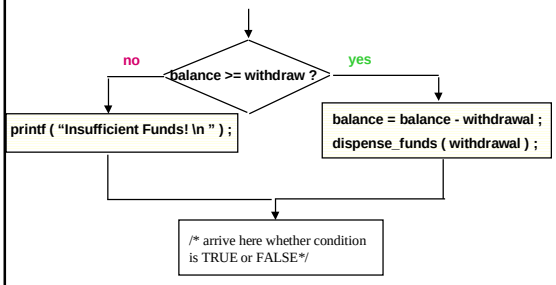
```
int abs ( int x )
{
  if ( x < 0 )
    x  =  – x ;
  return ( x ) ;
}
```


if - else

Print error message:

```
if ( balance >= withdrawal ) {
   balance = balance - withdrawal ;
   dispense_funds ( withdrawal ) ;
}
else {
   printf ( "Insufficient Funds! \n " ) ;
}
```


if - else Control Flow



Flow of Control

```
bal = 25.0 ;
withdrawal = 20.0 ;
if ( bal >= withdrawal ) {
   bal = bal - withdrawal ;
   dispense ( withdrawal ) ;
} else {
   printf("Insufficient...") ;
}
eject_card ( ) ;
```

```
bal = 25.0 ;
withdrawal = 40.0 ;
if ( bal >= withdrawal ) {
   bal = bal - withdrawal ;
   dispense ( withdrawal ) ;
} else {
   printf("Insufficient...") ;
}
eject_card ( ) ;
```


Formatting *if* statements

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

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

```
if (condition) {
   statement t1;
   statement t2;
   ...
}
```

```
if (condition) {
   statement t1;
   statement t2;
   ...
} else {
   statement e1;
   statement e2;
   ...
}
```


Alternative Formatting of *if*s

```
if (condition)
{
   statement t1;
   statement t2;
   ...
}
```

```
if (condition)
{
   statement t1;
   statement t2;
   ...
}
else
{
   statement e1;
   statement e2;
   ...
}
```


Nested *if*s

```
#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 *if*s , 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 ") ;
```


Dangling *else*

```
if ( x == 5 )
   if  ( y == 5 )
      printf ("Both are 5. \n") ;
else                                  /* misleading */
   printf ("Is anybody 5?\n") ;       /* indentation */
```

```
if ( x == 5 ) {
   if  ( y == 5 )
      printf ("Both are 5. \n") ;
   else
      printf ("Is anybody 5?\n") ; /*x is 5, y is not */
}
```

Matching *else*s with *if*s

- Each *else* matches some *if* in the same block

  if { if if else { if if else else } else } else

- Within the same block read the *if*s and *else*s left to right matching each *else* to the closest unmatched *if*

  if { if if else { if if else else } else } else

- Some *if*s may not be matched to any *else*

  if if if else else

  if if else if

Tax Example

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.

The First Character

```
/* 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 *first_character*

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

Sort 2 Characters: Top Down Design

Input two characters

Rearrange them in sorted order

Output them in sorted order

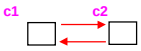
```
Input: ra      Output: ar
Input: nt      Output: nt
```


Sort 2 Characters: Refinement

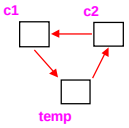
Input c1, c2
If c2 comes before c1 in alphabet
  Swap c1 and c2
Output c1, c2

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 Program

```
/* 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")

Complex Conditionals

- AND (&&), OR (||), NOT (!)
- Review: like arithmetic expressions, conditional expressions have a value:
  - TRUE or FALSE (non-zero or 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) ;* /* m is 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 ;
```

```
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 expression

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,

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

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

Proof of DeMorgan

Is is 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					

Operator Precedence

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 ) ;        /* no action or "null statement" */
    printf( "x  >  10 " ) ; /* Always done (see why?) */
```

Recall that any non-zero integer value is "true".

```
if ( x ) printf ( "x is nonzero" ) ;  /*works, but bad style */
```

```
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 ||

Beware == and != with doubles:

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


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:
```


Pitfalls 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 ) ;
```


switch on char

```
char marital_status ;
printf ( "Enter marital status (M,S): " ) ;
scanf ( "%c", &marital_status ) ;
switch ( marital_status ) {
case 'm':
case 'M':
    printf ( "Married \n" ) ;
    break ;
case 's':
case 'S':
    printf ( "Single \n" ) ;
    break ;
default:
    printf ( "Sorry, I don't recognize that code. \n" ) ;
}
```

int or char expression

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