CSE 142
Computer Programming I

Complex Conditions

From Homework Descriptions to Programs

Overview

Concepts this lecture

- **Part A**
  - **Not**
  - **Truth tables**
  - **DeMorgan's laws**
- **Part B**
  - **Design**
  - **Testing**

Boolean Operators in C

Conditionals often involve words like AND, OR, and NOT.

The Boolean operators AND, OR, and NOT have these symbols in C:

&&	||	!
and	or	not

Conditionals in C

if I have at least $15 or you have at least $15, then we can go to the movies:

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

if the temperature is below 32 degrees and it's raining, then it's snowing:

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


Assignment of Conditional Expressions

Sometimes it's more convenient (or clearer) to assign the value of a complex conditional to a variable than to write the conditional as a test.

The simplest case: TRUE and FALSE

```
int   TRUE = 1;
int   FALSE = 0;

int   canGoToMovies;
canGoToMovies  = TRUE ;
…
if (canGoToMovies) {
    …
}
```


Assignment of Conditional Expressions

```
int   TRUE = 1;
int   FALSE = 0;

int   canGoToMovies;

canGoToMovies  = myMoney>=15.0 || yourMoney>=15.0;
…
if (canGoToMovies) {
    …
}
```

Negating Conditions

Suppose we want a while loop to terminate as soon as *either x is 17 or x is 42*

Which is it?

while (x!=17 || x!=42) …

while (x!=17 && x!=42) …

Either way? Something else?

Truth tables and DeMorgan's Law give us tools for answering such questions

I-7

Truth Tables for && and ||

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 ("boolean value")

I-8

Truth Table for not (!)

P	!P
T	F
F	T

I-9

not (!) Example

```
int highRisk ;
highRisk =  age < 25  &&  sex  ==  'M'  ;
if ( highRisk ) {      /* Do nothing */
} else {
    printf ( "Cheap rates. \n") ;
}
```

P	!P
T	F
F	T

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

I-10

Equivalence of Complex Expressions

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

is equivalent to

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

Or is it?

I-11

DeMorgan's Laws

DeMorgan's laws help determine when two complex conditions are equivalent

They state:

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

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

This applies for any Boolean expressions P and Q, which might themselves be complex expressions

I-12

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	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

Exercise: Prove the other law

Solution To a Previous Question

We wanted a while loop to terminate as soon as either x is 17 or x is 42.

So the loop condition is
while (! (x==17 || x==42)) …
Using DeMorgan's Law we can rewrite as
while (x != 17 && x != 42) …

A truth table would show that
while (x != 17 || x != 42)
is wrong! (It's always true, for one thing…)

DeMorgan's Law Summary

(A great deal more notation is required to be 100% general and 100% correct, but this picture might help.)

!(……..)

||	⇒	&&
&&	⇒	||
!	⇒	
	⇒	!
<	⇒	>=
==	⇒	!=
>=	⇒	<

!(a<b || (!c && d==e))

↓

(a>=b && (c || d!=e))

(Warning: && has higher precedence than ||. Use (..)'s!)

Part B

Loop Development and Program Schemas

Goals for Loop Development

Getting from problem statement to working code

Systematic loop design and development

Recognizing and reusing code patterns

Example: Rainfall Data

General task: *Read daily rainfall amounts and print some interesting information about them.*

Input data: Zero or more numbers giving daily rainfall followed by a negative number (sentinel).

Example input data:

0.2 0.0 0.0 1.5 0.3 0.0 0.1 -1.0

Including the empty input sequence:

-1.0

Given this raw data, what sort of information might we want to print?

Rainfall Analysis

Some possibilities:

Just print the data for each day

Compute and print the answer to one of these questions

- *How many days worth of data are there?*
- *How much rain fell on the day with the most rain?*
- *On how many days was there no rainfall?*
- *What was the average rainfall over the period?*
- *What was the median rainfall (half of the days have more, half less)?*
- *On how many days was the rainfall above average?*

What's similar about these? Different?

Example: Print Rainfall Data

```
#include <stdio.h>
int main (void) {
    int SENTINEL = -1.0;
    double rain;          /* current rainfall from input */
    /* read rainfall amounts and print until sentinel (<0) */
    do  {
       rain = ReadDouble();
       if (rain != SENTINEL) {
          printf("%f  ", rain);
       }
    } while (rain >= 0.0);
    return 0;
}
```


Example: # Days in Input

```
#include <stdio.h>
int main (void) {
   int SENTINEL = -1.0;
   double rain;          /* current rainfall from input */
   int ndays = 0;            /* number of days of input */

    do {
       rain = ReadDouble();
       if (rain > 0.0) {
          ndays = ndays + 1;
       }
   } while (rain != SENTINEL);
   printf("# of days input = %d.\n", ndays);
   return 0;
}
```


Is There a Pattern Here?

```
#include <stdio.h>
int main (void) {
    int SENTINEL = -1.0;
    double rain;
    do  {
       rain = ReadDouble();
       If (rain != SENTINEL) {
          printf("%f  ", rain);
       }
    } while (rain != SENTINEL);
    return 0;
}
```

```
#include <stdio.h>
int main (void) {
   int SENTINEL = -1.0;
   double rain;
   int ndays = 0;

    do {
       rain = ReadDouble();
       if (rain > 0.0) {
          ndays = ndays + 1;
       }
   } while (rain != SENTINEL);
   printf("# of days input = %d.\n", ndays);
   return 0;
}
```


Program Schema

A program schema is a pattern of code that solves a general problem

Also called a "design pattern"

Learn patterns through experience, observation.

If you encounter a similar problem, try to reuse the pattern

Tips For Problem Solving

Given a problem to solve, look for a familiar pattern

Work the problem by hand to gain insight into possible solutions. Ask yourself "what am I doing?"

Check your code by hand-tracing on simple test data.