Course Notes Set 3:

COMP1200-001

Introduction to Computing for Engineers and Scientists

C Programming

Control Structures: Selection

Computer Science and Software Engineering

Auburn University

Overview

- Precedence Rules Review
- Logical Operators
- Conditional Expressions
- Selection Statements
- Repetition
- Algorithm Development
- Data Files

Precedence Rules Review

1. Parenthesis. Inner to outer.	
2. Unary Operators +, -, ++, --.	Right to left.
3. Binary Operators *, /, and %.	Left to right.
4. Binary Operators + and -.	Left to right.
5. Assignment Operators =, +=, -=, *=, /=, %=.	Right to left.

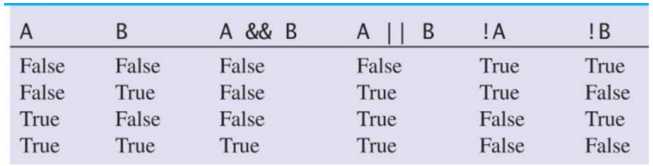
Logical Operators

Logical operators

Must return TRUE or FALSE!

Operator	Name	Operation	Operator type
!	NOT	Negation	Unary
&&	AND	Conjunction	Binary
||	OR	Inclusive disjunction	Binary

Logical Operators
Boolean Logic is fun!



A	B	A && B	A || B	!A	!B
False	False	False	False	True	True
False	True	False	True	True	False
True	False	False	True	False	True
True	True	True	True	False	False

Etter, Engineering Problem Solving with C, Third Edition, © 2005 Pearson Education, Inc. All rights reserved. 0-13-142971-X

Relational operators

Relational operator	Meaning
<	less than
<=	less than or equal to
==	equal to
>	greater than
>=	greater than or equal to
!=	not equal to

Warnings about “==“

== with float or double type

Avoid using == when comparing a float or double type variable

Use >= or <= to “catch the almost equal to

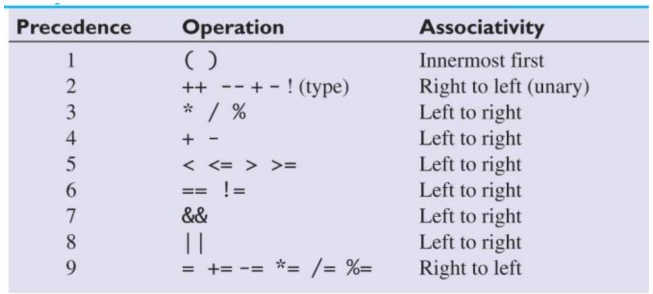
double balance = 0.0000000001;

balance is very close to but not equal to 0.0

COMMON ERROR

x == 1 IS NOT THE SAME as x = 1

Precedence for Arithmetic, Relational, and Logical Operators



Precedence	Operation	Associativity
1	( )	Innermost first
2	++ -- + - ! (type)	Right to left (unary)
3	* / %	Left to right
4	+ -	Left to right
5	< <= > >=	Left to right
6	== !=	Left to right
7	&&	Left to right
8	||	Left to right
9	= += -= *= /= %=	Right to left

Etter, Engineering Problem Solving with C, Third Edition, © 2005 Pearson Education, Inc. All rights reserved. 0-13-142971-X

NOT (!) Relation operator

Let **x = 7, y = 8**;

```
!(x>5)                    FALSE
!(y<x)                    TRUE

!((x>5)||(y<10))
!(TRUE || TRUE )
!(TRUE)                   FALSE

!((x>5)&&(y<8))
!(TRUE && FALSE)
!(FALSE)                  TRUE
```

Precedence example

```
a=4, b=-2, c=0

X = ( a > b || b > c && a == b )
X = ( 4 >-2 ||-2 > 0  && 4 ==-2 )
X = ( TRUE  || FALSE && FALSE  )
X = ( TRUE  ||       FALSE     )
X = (      TRUE                )
```

Control Sructures

Control Structures

Simple sequential C program

- Read information
- Calculate information
- Print information

Most solutions to problems require more complicated steps.

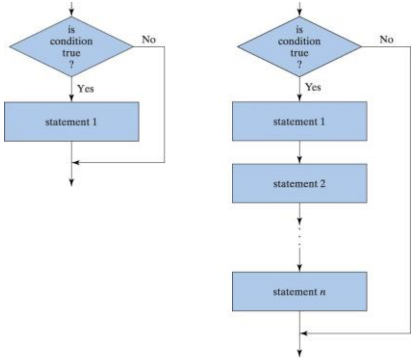
Control Structures

- Top-Down Design
  - big picture
  - sequential steps
- Decomposition Outline
  - simple problems
    - list steps
  - complicated problem
    - divide and conquer
      - stepwise refinement

Control Structures

- Structured Programming
  - Simple control structures
    - Sequence (ex. A long math formula)
      - one after another
    - Selection (ex. Taxable or not)
      - condition
    - Repetition (ex. Reading a file until the end)
      - loop

Flowcharts for Selection Statements: Abstract Example

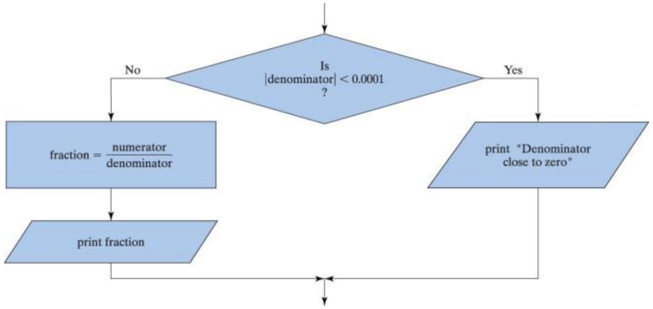


Etter, Engineering Problem Solving with C, Third Edition, © 2005 Pearson Education, Inc. All rights reserved. 0-13-142971-X

Simple if statement

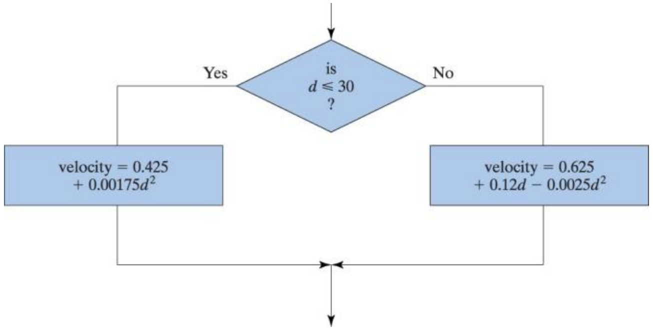
```
if( taxable == “yes” )       //condition
 { //body, executed if true
    price = price + price * 0.08;
 } //end of if statement

 //statements always executed
 subtotal = subtotal + price;
```

Flowchart for Selection Structure


Is
|denominator| < 0.0001
?
No
Yes
fraction = numerator / denominator
print "Denominator close to zero"
print fraction

Etter, Engineering Problem Solving with C, Third Edition, © 2005 Pearson Education, Inc. All rights reserved. 0-13-142971-X

Flowchart for `if/else` Statement


is
d ⩽ 30
?
Yes
No
velocity = 0.425 + 0.00175d²
velocity = 0.625 + 0.12d − 0.0025d²

Etter, Engineering Problem Solving with C, Third Edition, © 2005 Pearson Education, Inc. All rights reserved. 0-13-142971-X

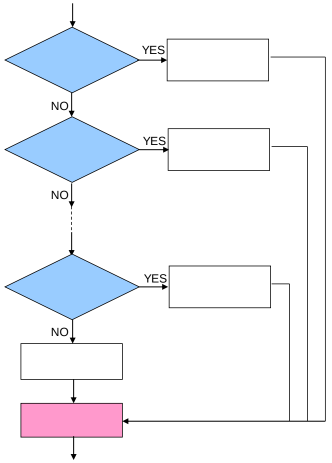
if-else

```
 if( d <= 30 ) //condition
{
   velocity = 0.425 + 0.00175 * d * d;
}
else          //when the condition fails
{
   velocity = 0.625 + 0.1* d + 0.0025 * d * d;
} //end of if/else

printf(“Velocity is %9.2f.\n”);
 . . .
```

```
if( hours > 40 ) //first condition
 {
   if( salaried == ‘Y’ ) //second condition
   {
     OT = 0;
   }
   else
   {
     OT = ( hours - 40 ) * 1.5 * wage;
   } //end of second if/else
 }
 else
 {
   OT = 0;
 } //end of first if/else
```

Nested-if-else

if-else-if
switch



```
#include <stdio.h>
int main()
{
   int month;
   int numDays;
   printf("Enter a month: ");
   scanf("%d", &month);

   if ( month == 2 )
      numDays = 28;

   else if ( month == 4 || month == 6 ||
             month == 11 || month == 9 )
      numDays = 30;

   else
      numDays = 31;

    printf("\nMonth %d has %d days.\n", month, numDays);
   return 0;
}
```

if-else-if

Switch Statements

- Acts as a “chooser” function
- Like an **`if-else-if`** statement
- How it works:
  - Take in a “choice” value
  - Go to a corresponding numbered “case”

```
#include <stdio.h>
int main()
{
   int month;
   int numDays;
   printf("Enter a month: ");
   scanf("%d", &month);
   switch (month)
   {
      case 2:  numDays = 28;
               break;
      case 4:
      case 6:
      case 9:
      case 11: numDays = 30;
               break;
      default: numDays = 31;
   }
   return 0;
}
```

switch

Traffic Fine Example

Sequential programs

A sequential program runs from beginning to end, one step at a time, executing every step in the program and repeating no steps.

Problem Statement

Write a program that computes a traffic fine. Suppose that the fine is $5 for every mile-per-hour over the speed limit a motorist is.

Determine Input/Outputs

To compute the fine we need to know:

The speed limit

The speed of the motorist

The output will be the fine in fractional dollars (two digits after the decimal point).

Sequential program - algorithm

Given:

speed limit = 45

speed of motorist = 50

fine = (speed of motorist - speed limit) * 5.0

Result:

fine = (50 - 45) * 5.0 = $25.0

Develop Algorithm

1. Get the speed limit
2. Get the motorist speed
3. Compute the fine
4. Output the fine

```
#include <stdio.h>
int main(void)
{
  /* Speed limit on this area of road */
  double   speedLimit;
  /* Speed the motorist was traveling */
  double   motoristSpeed;
  /* Computed fine */
  double   fine;
```

```
  /* Get speed limit */
  printf("Enter speed limit:");
  scanf("%lf",&speedLimit);

  /* Get motorist speed */
  printf("Enter motorist speed:");
  scanf("%lf",&motoristSpeed);

  /* Compute fine */
  fine=(motoristSpeed-speedLimit)*5.0;

  /* Output fine */
  printf("Fine = $%.2f\n",fine);

  return 0;
}
```

Running this program would result in the following sample session:

```
Enter speed limit: 45
Enter motorist speed: 50
Fine = $25.00
```

BUT there is a problem with this program.

What if an error occurs and the user of the program puts in a motorist speed that is lower than the speed limit?

```
Enter speed limit: 45
Enter motorist speed: 40
Fine = $-25.00
```

Conditional statements

We should only compute the fine if the input is good. To do this, we need to modify our algorithm slightly:

```
1. Get the speed limit
2. Get the motorist speed
3. If motorist speed is greater than speed limit then
   3.1 Compute the fine
   3.2 Output the fine
   printf("Enter speed limit:");    //Get speed limit
   scanf("%lf",&speedLimit);

   printf("Enter motorist speed:"); //Get motorist speed
   scanf("%lf",&motoristSpeed);

   if (motoristSpeed > speedLimit)  //is this input
                                    // correct?
   {
      fine=(motoristSpeed - speedLimit)* 5.0;
      printf("Fine = $%.2f\n",fine);
   }

   return 0;
}
```

if

Shouldn’t we say something, if the input is bad?
To do this, we need to print an error message.
Let’s modify the program again:

```
     printf("Enter speed limit:"); //get speed limit
     scanf("%lf",&speedLimit);

     printf("Enter motorist speed:"); //get motorist
     speed
     scanf("%lf",&motoristSpeed);

      //is this input correct?

     if (motoristSpeed > speedLimit)
     {
        fine=(motoristSpeed - speedLimit)* 5.0;
        printf("Fine = $%.2f\n",Fine);
     }
     else
     {
        printf("Error: Speed limit too large\n");
     }
     return 0;
}
```

if-else

What if the fine is determined by a region code.

Region 1	$10.25 per mile over
Region 2	$7.50 per mile over
Others	$5.00 per mile over

We’ll need to add an input statement to read in the region code, then we’ll need to test the region code to see what fine rate to charge. We could write an *if* statement like this:

```
if (region == 1)
{
   fineRate = 10.25;
}
else if (region == 2)
{
   fineRate = 7.50;
}
else
{
   fineRate=5.00;
}
```

if-else-if

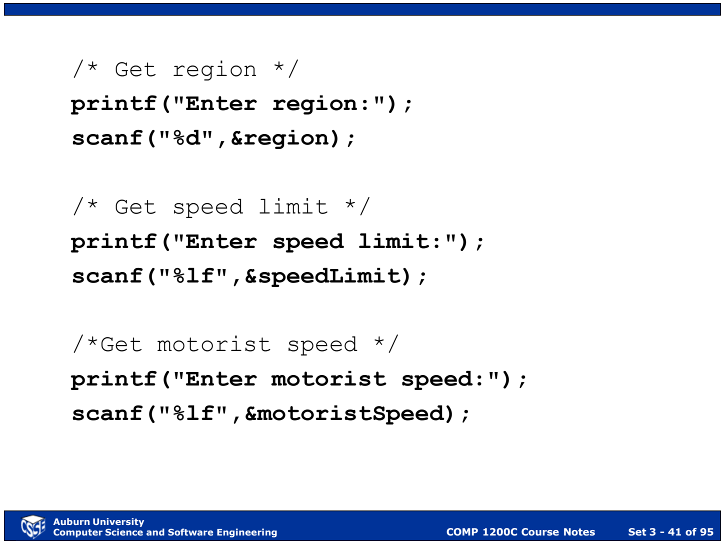
This form of the *if* statement is called an *if-else-if* construct.

There is another condition statement, however, called the *switch* that is more appropriate.

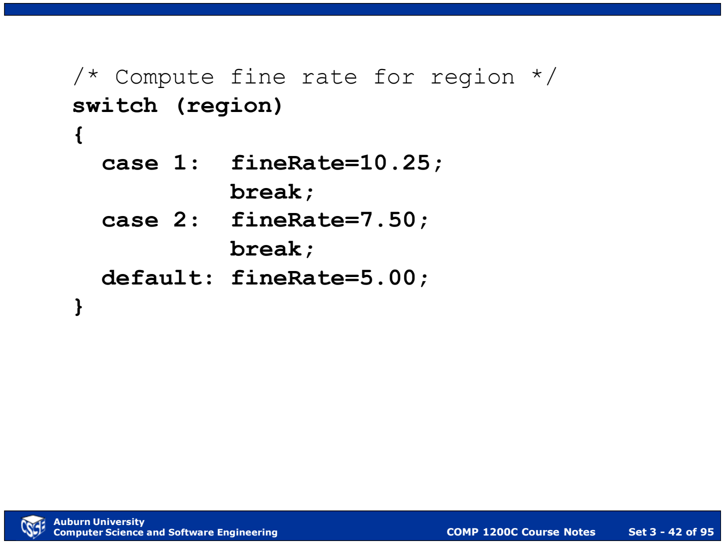
```
switch (region)
{
  case 1:
          fineRate=10.25;
          break;
  case 2:
          fineRate=7.50;
          break;
  default:
          fineRate=5.00;
}
```

switch

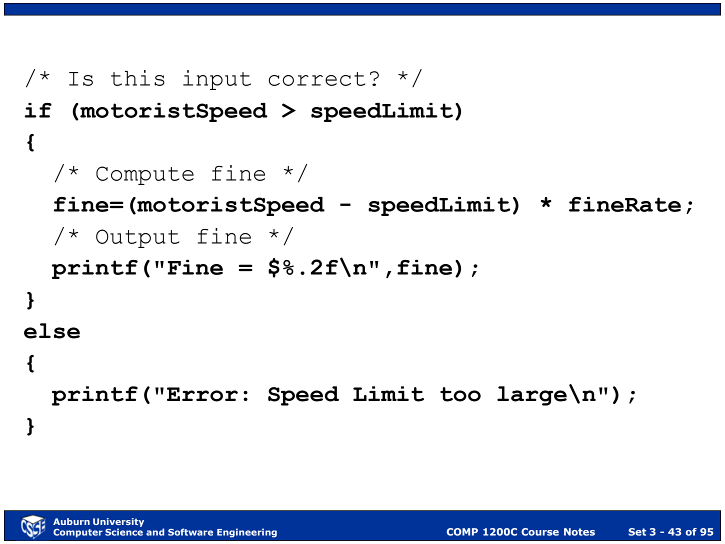
```
#include <stdio.h>
int main(void)
{
   double speedLimit;    //speed limit
   double motoristSpeed; //motorist speed
   double fine;    //computed fine
   int    region; //region infraction occurred

   double fineRate;      // region fine
```



```
/* Get region */
printf("Enter region:");
scanf("%d",&region);

/* Get speed limit */
printf("Enter speed limit:");
scanf("%lf",&speedLimit);

/*Get motorist speed */
printf("Enter motorist speed:");
scanf("%lf",&motoristSpeed);
```



```
/* Compute fine rate for region */
switch (region)
{
  case 1:  fineRate=10.25;
           break;
  case 2:  fineRate=7.50;
           break;
  default: fineRate=5.00;
}
```



```
/* Is this input correct? */
if (motoristSpeed > speedLimit)
{
  /* Compute fine */
  fine=(motoristSpeed - speedLimit) * fineRate;
  /* Output fine */
  printf("Fine = $%.2f\n",fine);
}
else
{
  printf("Error: Speed Limit too large\n");
}
```