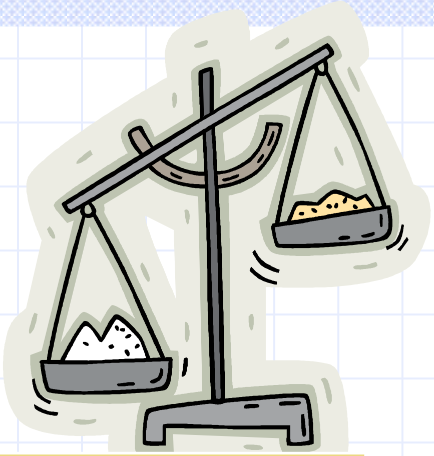
ESC101: Introduction to Computing

Conditional Expressions

Relational Operators



- Compare two quantities

Operator	Function
>	Strictly greater than
>=	Greater than or equal to
<	Strictly less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

- Work on int, char, float, double…

Examples

Rel. Expr.	Result	Remark
3>2	1	
3>3	0	
‘z’ > ‘a’	1	ASCII values used for char
2 == 3	0	
‘A’ <= 65	1	'A' has ASCII value 65
‘A’ == ‘a’	0	Different ASCII values
(‘a’ – 32) == ‘A’	1	
5 != 10	1	
1.0 == 1	AVOID	May give unexpected result due

Avoid mixing int and float values while comparing. Comparison with floats is not exact!

Logical Operators

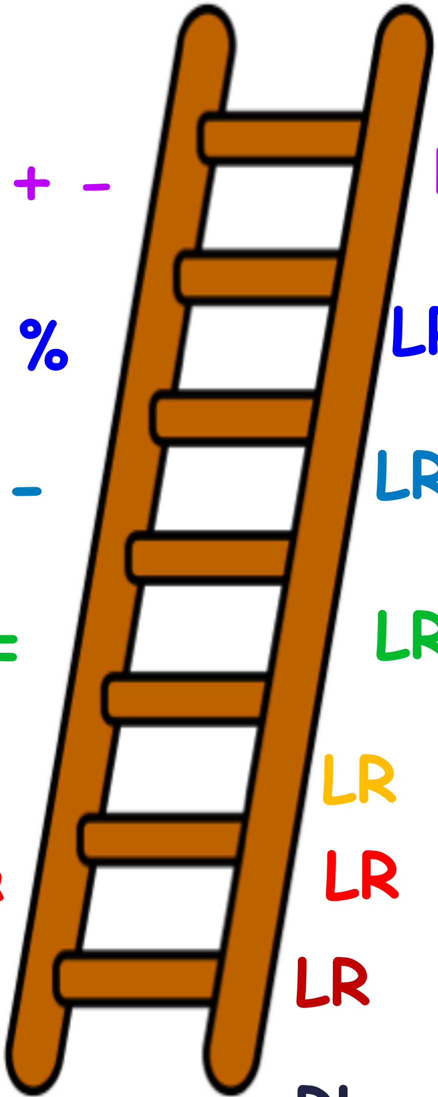
Logical Op	Function	Allowed Types
&&	Logical AND	char, int, float, double
||	Logical OR	char, int, float, double
!	Logical NOT	char, int, float, double

Remember
- value 0 represents false.
- any other value represents true.

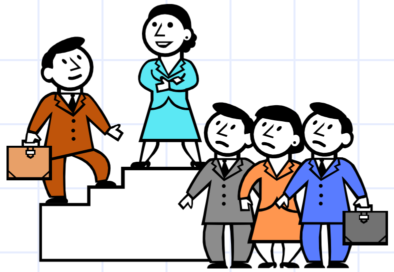
Examples

Expr	Result	Remark
2 && 3	1	
2 || 0	1	
‘A’ && 0	0	
‘A’ && ‘0’	1	ASCII value of ‘0’≠0
‘A’ && ‘b’	1	
! 0.0	1	0.0 == 0 is **guaranteed**
! 10.05	0	Any real ≠ 0.0
(2<5) && (6>5)	1	Compound expr

Precedence and Associativity (Refined)



! + -	RL
* / %	LR
+ -	LR
< <= > >=	LR
== !=	LR
&&	LR
||	LR
=	RL



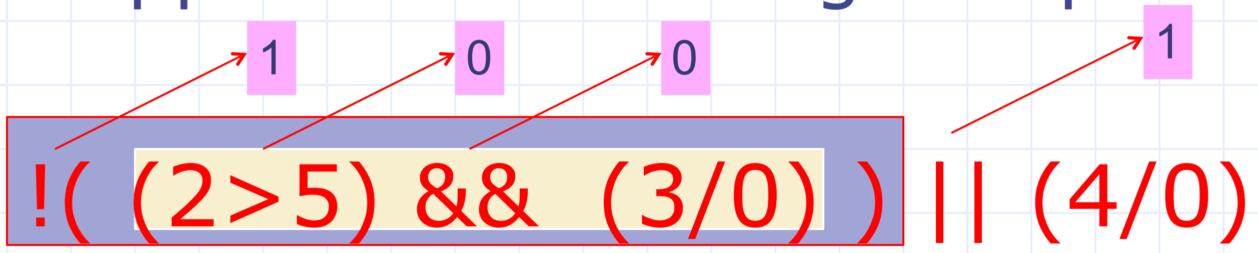
Order of evaluation

- Logical operators && and || guarantee evaluation of operands from left to right.

- They evaluate the smallest number of operands: short-circuit evaluation

Short-circuit Evaluation

- Do not evaluate the second operand of binary logical operator if result can be deduced from first operand
  - Arguments of && and || are evaluated from left to right (in sequence)
  - Also applies to nested logical operators



Evaluates to 1

Expression evaluation

- Precedence
  - Applied to two different class of operators
  - + and *, - and *, && and ||, + and &&, …
- Associativity
  - Applied to operators of same class
  - * and *, + and -, * and /, …
- Order of evaluation
  - Precedence and associativity identify the operands for each operator (Parenthesization)

Conditional statements in C

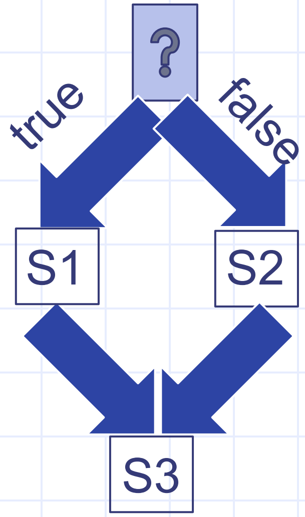
- Three types of conditional statements in C
  - if (condition) *action*

    else *some-other-action*
  - if (condition) *action*
  - switch-case
- Each action is a sequence of one or more statements

if-else statement

- General form of the if-else statement

```
if (expression)
        statement S1
else
        statement S2
statement S3
```

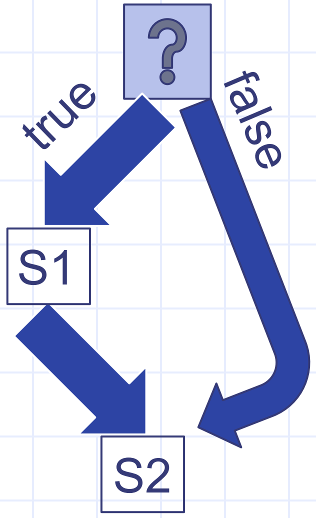


- Execution of if-else statement
  - First the expression is evaluated.
  - If it evaluates to a non-zero value, then S1 is executed and then control (program counter) moves to S3.
  - If expression evaluates to 0, then S2 is executed and then control moves to S3.
  - S1/S2 can be **block** of statements

if statement (no else!)

- General form of the if statement

```
if (expression)
        statement S1
statement S2
```



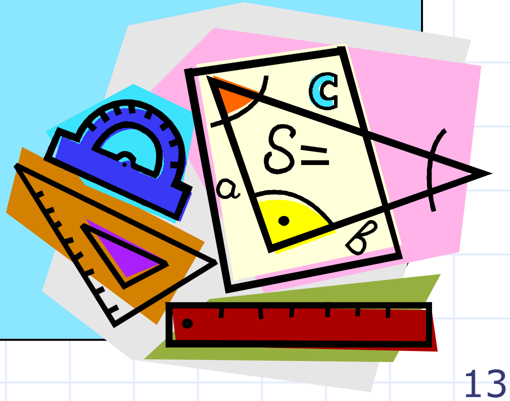
- Execution of if statement
  - First the expression is evaluated.
  - If it evaluates to a non-zero value, then S1 is executed and then control (program counter) moves to the statement S2.
  - If expression evaluates to 0, then S2 is executed.

Example

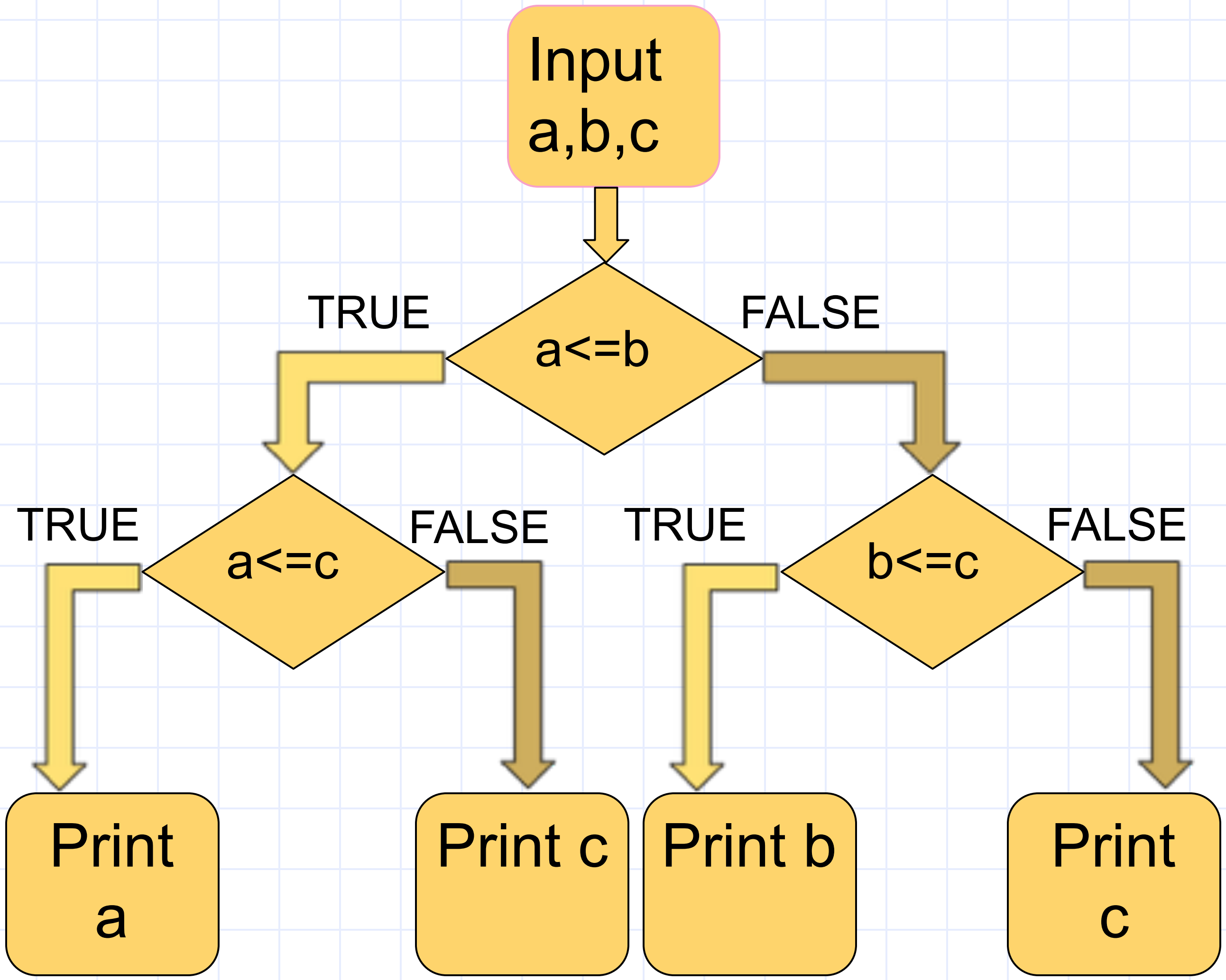
- Problem: Input a, b, c are real positive numbers such that c is the largest of these numbers. Print ACUTE if the triangle formed by a, b, c is an acute angled triangle and print NOT ACUTE otherwise.

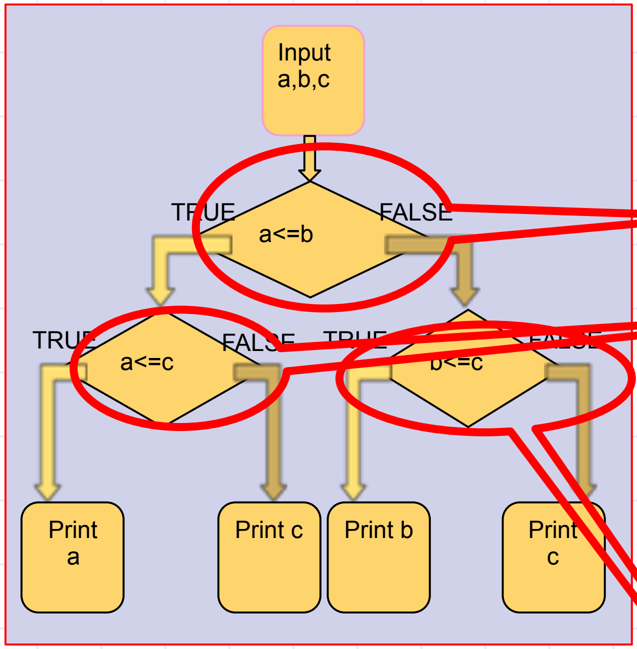
```
int main() {
        float a; float  b; float  c;
                scanf("%f%f%f", &a,&b,&c);          /* input a,b,c */

        if ( (a*a + b*b) >  (c*c) )   {  /* expression*/
                printf("ACUTE");
            }
            else {
                printf("NOT ACUTE");
            }
        return 0;
    }
```



Finding the minimum of three numbers





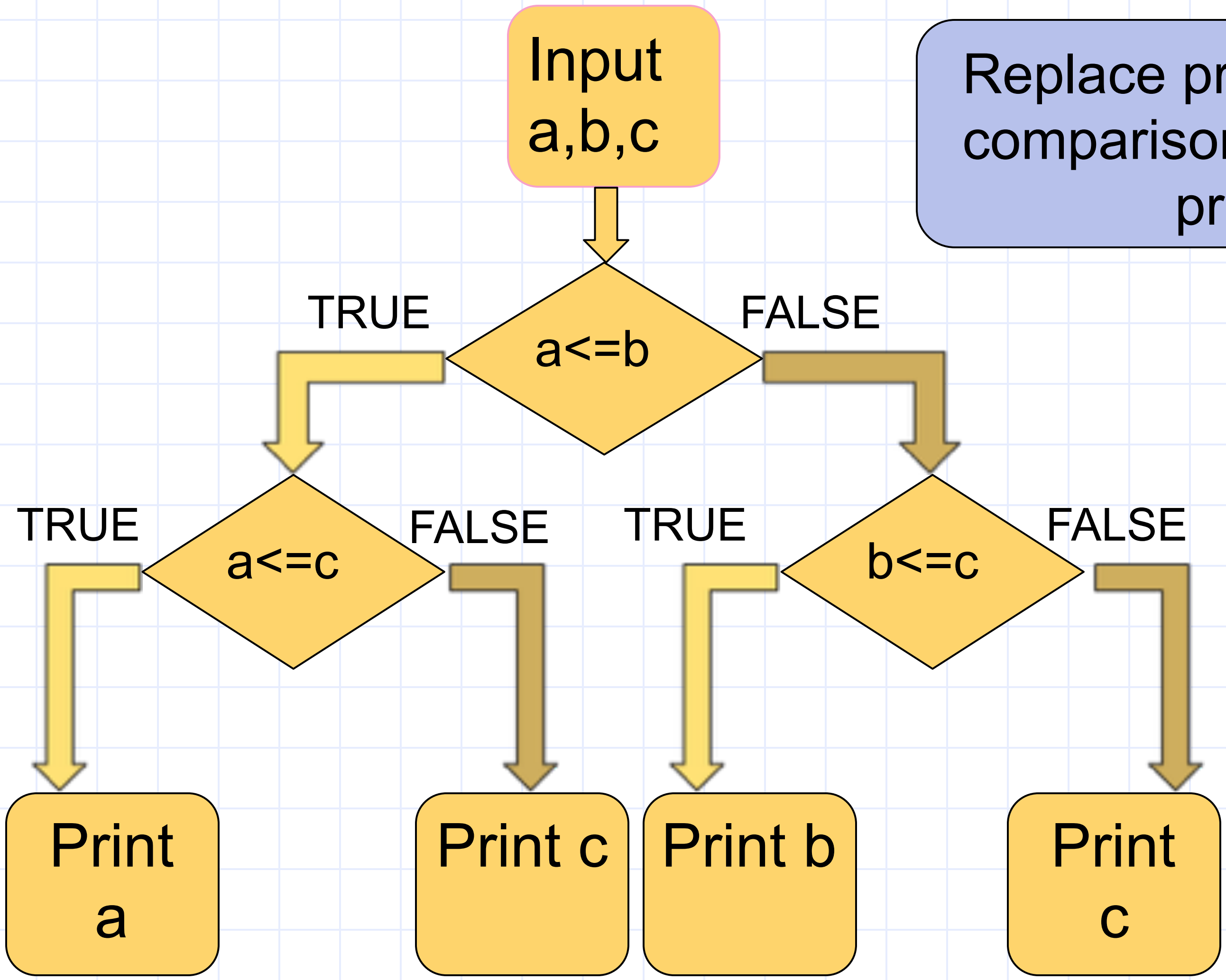
- Each branch translates to an if-else statement
- Hierarchical branches result in nested if statements

```
int a,b,c;
scanf(“%d%d%d”,&a,&b,&c);
if (a <= b) {
      if (a <= c) {
            printf(“min = %d”,a);
      }
      else {
            printf(“min = %d”, c);
      }
}
else {
       if (b <= c)  {
             printf(“min = %d”, b);
       }
       else {
             printf(“min =%d”, c);
       }
}
```

More Conditionals

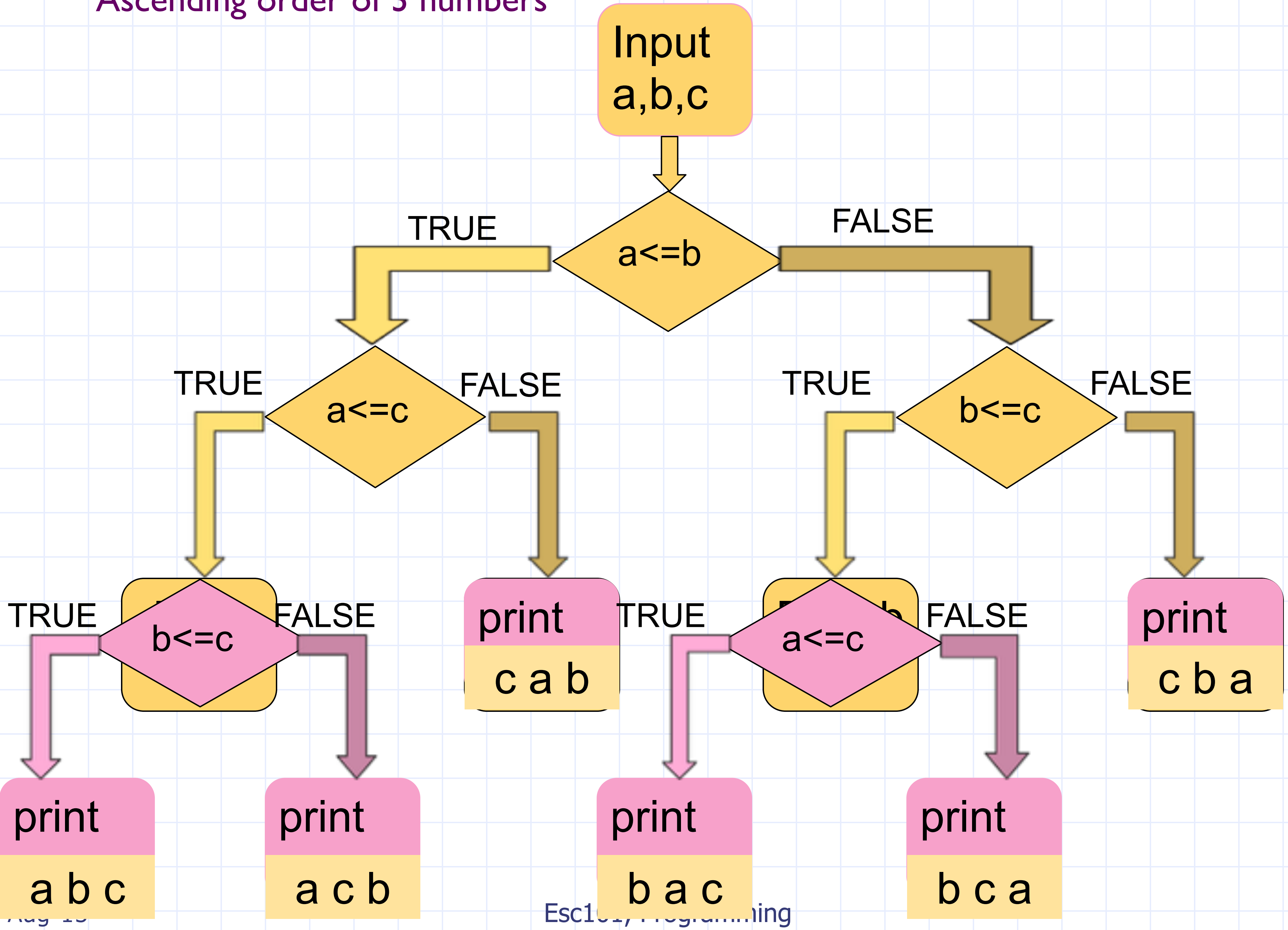
- Sorting a sequence of numbers (i.e., arranging the numbers in ascending or descending order) is a basic primitive.
- Problem: read three numbers into a, b and c and print them in ascending order.
  - Start with the flowchart for finding minimum of three numbers and add one more level of conditional check.
  - Then translate the flowchart into C program.

Finding min of 3 numbers



Replace print by more comparisons and then print.

Ascending order of 3 numbers



```
if (a <= b) {
    if (a <= c) {          /* a <= b and a <= c */
           if (b <= c) {   /* a <= b, a <= c, b <= c */
            printf(“%d %d %d \n”, a, b, c);
           } else {        /* a <= b, a <= c, c < b */
               printf(“%d %d %d \n”, a, c, b);
           }
    } else {               /* a <= b, c < a*/
           printf(“%d %d %d \n”, c, a, b) ;
        }
} else {                   /* b < a */
    if (b <= c) {         /* b < a and b <= c */
           if (a <= c) {  /* b < a, b <= c, a <= c */
            printf(“%d %d %d\n”, b, a, c);
        } else {           /* b < a, b <= c, c < a */
               printf(“%d %d %d\n”, b, c, a); }
        }
    } else {              /* b < a, c < b */
        printf(“%d %d %d\n”, c, b, a); }
    }
}
```

Changing a capital to small character and vice versa

```
# include <stdio.h>
int main(){
  char c;
  scanf("%c",&c);  // assume valid character
  if(  c >= 'a' && c <='z' )
      c = c -'a' + 'A';
   if ( c >= 'A' && c <='Z' )
      c = c - 'A' + 'a';
   printf("Char is now %c\n",c);
   return 0;
 }
```

Input 'X'

Output: Char is now x

Changing a capital to small character and vice versa

```
# include <stdio.h>
int main(){
  char c;
  scanf(“%c”,&c);  // assume valid character
  if(  c >= ‘a’  && c <=‘z’ )
      c = c -‘a’ + ‘A’;
   if ( c >= ‘A’  && c <=‘Z’ )
      c = c - ‘A’ + ‘a’;
   printf(“Char is now %c\n”,c);
   return 0;
}
```

Input ‘x’ **Incorrect** Output: Char is now x

Changing a capital to small character and vice versa

```
# include <stdio.h>
int main(){
  char c;
  scanf("%c",&c);  // assume valid character
  if(  c >= 'a' && c <='z' )
      c = c -'a' + 'A';
   else {
      if ( c >= 'A' && c <='Z' )
         c = c - 'A' + 'a';
   }
   printf("Char is now %c\n",c);
   return 0;
}
```

Input 'x' **Correct** Output: Char is now X

Nested if, if-else

- Earlier examples showed us *nested* if-else statements

```
if (a <= b) {
        if (a <= c) { … }  else {…}
} else {
        if (b <= c)  { … } else { … }
}
```

- Because if and if-else are also statements, they can be used anywhere a statement or block can be used.

Else if

- A special kind of nesting is the chain of if-else-if-else-… statements

```
if (cond1) {
     stmt1
} else {
        if (cond2)  {
             stmt2
        } else {
            if (cond3) {
                ….
            }
        }
}
```

General form of if-else-if-else…

```
if (cond1)
     {stmt-block1}
else if (cond2)
     {stmt-block2}
else if (cond3)
     {stmt-block3}
else if (cond4)
     {stmt-block4}
else if  …
else
      last-block-of-stmt
```

Example

- Given an integer day , where $1 \leq day \leq 7$, print the name of the weekday corresponding to day.

  1: Sunday

  2: Monday

  ...

  7: Saturday

Printing the day

```
int day;
scanf (“%d”, &day);
if (day == 1) { printf(“Sunday”); }
else if (day == 2) { printf (“Monday”); }
else if (day == 3) { printf (“Tuesday”); }
else if (day == 4) { printf (“Wednesday”); }
else if (day == 5) { printf (“Thursday”); }
else if (day == 6) { printf (“Friday”); }
else if (day == 7) { printf (“Saturday”); }
else { printf (“ Illegal day %d”, day); }
```

Example 2

- Given an integer day , where $1 \leq day \leq 7$, print Weekday, if the day corresponds to weekday, print Weekend otherwise.

  1, 7: Weekend

  2,3,4,5,6: Weekday

Weekday - version 1

```
int day;
scanf (“%d”, &day);
if (day == 1) { printf(“Weekend”); }
else if (day == 2) { printf (“Weekday”); }
else if (day == 3) { printf (“Weekday”); }
else if (day == 4) { printf (“Weekday”); }
else if (day == 5) { printf (“Weekday”); }
else if (day == 6) { printf (“Weekday”); }
else if (day == 7) { printf (“Weekend”); }
else { printf (“ Illegal day %d”, day); }
```

Weekday - version 2

```
int day;
scanf (“%d”, &day);
if ((day == 1) || (day == 7)) {
        printf(“Weekend”);
} else if (  (day == 2) || (day == 3)
              || (day == 4) || (day == 5)
              || (day == 6)) {
        printf (“Weekday”);
} else {
        printf (“ Illegal day %d”, day);
}
```

Weekday - version 3

```
int day;
scanf (“%d”, &day);
if ((day == 1) || (day == 7)) {
        printf(“Weekend”);
} else if ( (day >= 2) && (day <= 6) ) {
        printf (“Weekday”);
} else {
        printf (“ Illegal day %d”, day);
}
```

Summary of if, if-else

- if-else, nested if's, else if.
- Braces {…} can be omitted if a block has only one statement.
- Multiple ways to solve a problem
  - issues of better readability
  - and efficiency.

if-else

```
if ((a != 0) && (b != 0))
    if (a * b >= 0)
        printf (“positive”);
else
    printf(“zero”);
```

OUTPUT for a = 5, b = 0

NO OUTPUT!!

OUTPUT for a = 5, b = -5

zero

```
if ((a != 0) && (b != 0))
    if (a * b >= 0)
        printf (“positive”);
    else
        printf(“negative”);
```

OUTPUT for a = 5, b = 0

NO OUTPUT!!

OUTPUT for a = 5, b = -5

negative

Unmatched if and else

- An else always matches closest unmatched if
  - Unless forced otherwise using { … }

```
if (cond1)
    if (cond2)
        …
   else
        …
```

```
if (cond1) {
    if (cond2)
        …
    else
        …
}
```

Unmatched if and else

- An else always matches closest unmatched if
  - Unless forced otherwise using { … }

```
if (cond1)
    if (cond2)
        …
  else
        …
```

IS NOT SAME AS

```
if (cond1) {
    if (cond2)
        …
}
else
        …
```

Next class

- Switch statement and loops