

## Conditional Statements

For computer to make decisions, must be able to test CONDITIONS

IF *it is raining*  
THEN *I will not go outside*

IF *Count is not zero*  
THEN *the Average is Sum divided by Count*

Conditions specified using logical data

## Outline

### II. Program Basics

#### G. Expressions

- 3. Binary operators
  - relational operators == != < > <= >=
  - logical operators && || !

#### 7. Logical data

#### H. Statements

- 3. If
- 4. If-Else
  - matching else
- 5. Switch
  - default
  - break

## Logical Data in C

- no explicit logical type
- int and char values can be treated as logical values
- two possible logical values
  - 0 or '\0' - false
  - anything else - true
- specific operators produce logical values

## Logical Expressions

- Relational operators:
  - operators to compare two values
  - syntax: *expr1 relop expr2*
  - expressions must be of same type (any)
- Logical operators:
  - operators to logically combine logical values
  - produce complex combinations of conditions
  - operators apply to logical values

## Relational Operators

Operator	Meaning
X == Y	X equal to Y
X != Y	X not equal to Y
X < Y	X less than Y
X > Y	X greater than Y
X <= Y	X less than or equal to Y
X >= Y	X greater than or equal to Y

## Relational Examples

A: 4 B: 3 C: 2.5

A < B 0 (false)  
'A' < 'B' 1 (true)

When chars compared, ASCII codes compared

'A' != 'a' 1 (true)

Note, expressions must be of same type  
But, implicit conversions will be done

B > C 1 (true), B converted to float

## Combined Expressions

Complex combinations of operations can be used:

4 < 3 + 2  
 4 < 5  
 1 (true)

Why isn't expression evaluated as (4 < 3) + 2?

Relational operators have lower precedence than arithmetic operators, higher than assignment

Have left-associativity

Operator	Precedence
> < >= <=	10
= !=	9

## Complex Example

```
10 % 4 * 3 - 8 <= 18 + 30 / 4 - 20
2 * 3 - 8 <= 18 + 30 / 4 - 20
6 - 8 <= 18 + 30 / 4 - 20
6 - 8 <= 18 + 7 - 20
-2 <= 18 + 7 - 20
-2 <= 25 - 20
-2 <= 5
1 (true)
```

## Logical Operators

- Simple (relational) logical expressions true or false depending on values compared
- Compound logical expressions built out of logical combinations of simple expressions
- Examples:
  - “I am standing in a lecture hall AND the moon is made of green cheese” is false, but
  - “I am standing in a lecture hall OR the moon is made of green cheese” is true

## Logical Expression

Syntax:

```
LogExpr1 LogicalOp LogExpr2 (&& ||)
LogicalOp LogExpr (!)
```

```
(2 < 3) && (17 != 29) /* && - AND */
1 && 1 /* true AND true */
1 /* true */
```

## AND (&&) and OR (||)

- AND (&&) of two logical expressions: resulting compound expression true when both expressions true
- OR (||) of two logical expressions: expression true when either expression true
- Truth table:

E1	E2	E1 && E2	E1    E2
1 (true)	1 (true)	1 (true)	1 (true)
1 (true)	0 (false)	0 (false)	1 (true)
0 (false)	1 (true)	0 (false)	1 (true)
0 (false)	0 (false)	0 (false)	0 (false)

## Logical Examples

```
month is 2, year is 1999
(month == 2) && ((year % 4) == 0)
1 (true) && ((year % 4) == 0)
1 && (3 == 0)
1 && 0 (false)
0 (false)

day is 6
(day == 0) || (day == 6)
0 (false) || (day == 6)
0 || 1 (true)
1 (true)
```

## Short-Circuit Aspect

- && and || are *short-circuit* operators - if the first operand determines the value, the second operand is not evaluated
- first operand of && is 0, result is 0 without evaluating second operand
- first operand of || is 1, result is 1 without evaluating second operand
- second operand evaluated when needed

## Negation

- Arithmetic negation:  
-(4 \* A \* C)  
Operator minus (-) negates numeric value
- Logical negation  
Logical operator (!) negates logical value  
E                    ! E  
1 (true)    0 (false)  
0 (false)   1 (true)  
Example: !(3 > 4)  
          ! 0 (false)  
          1 (true)

## Complex Expressions

Expressions can be built using arithmetic, relational and logical operators

```
!((-5 >= -6.2) || (7 != 3) && (6 == (3 + 3)))
```

<u>operator</u>	<u>precedence, associativity</u>
+ - (unary) ! & *	15 right
(typename)	14 right
* / %	13 left
+ - (binary)	12 left
< > <= >=	10 left
== !=	9 left
&&	5 left
	4 left
assignments	2 right

## Example

```
!((-5 >= -6.2) || (7 != 3) && (6 == (3 + 3)))  
!( 1 (true)    || (7 != 3) && (6 == (3 + 3)))  
!( 1            || 1 (true) && (6 == (3 + 3)))  
!( 1            || 1            && (6 == 6    ))  
!( 1            || 1            && 1 (true)    )  
!( 1            ||               1 (true)    )  
!  
          1 (true)  
0 (false)
```

## Decision Making

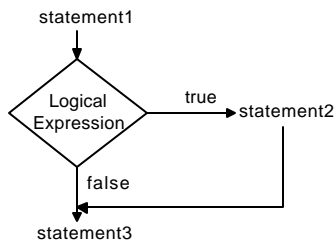
- Problem: input two numbers and store the largest in Big  
Num1: 17 Num2: 53 Big: 53
- Algorithm
  1. Input Num1 and Num2
  2. Store largest of Num1 and Num2 in Big
    - 2.1 Store Num1 in Big
    - 2.2 IF Num2 > Num1 THEN store Num2 in Big

## IF Statement

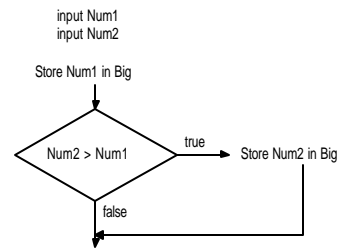
- Syntax: `if (LogicalExpr) Statement`
- Program context:  
statement1;  
if (LogicalExpr) statement2;  
statement3;
- Order of execution:  

<u>LogicalExpr</u>	
1 (true)	0 (false)
statement1	statement1
statement2	statement3
statement3	

### Flow of Execution



### Flow Chart for our Problem



### Code for Solution

```

int findMax(int Num1, int Num2) {
    int Big;

    Big = Num1;
    if (Num2 > Num1) Big = Num2;
    return Big;
}
  
```

### Trace of Solution

Trace of findMax(17,53):

	Num1	Num2	Big
statement	17	53	?
Big = Num1;			17
if (Num2 > Num1)			
Big = Num2;			53
return Big;			

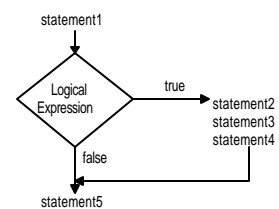
### Trace of Solution

Trace of findMax(9,6):

	Num1	Num2	Big
statement	9	6	?
Big = Num1;			9
if (Num2 > Num1)			
return Big;			

### Executing > 1 Statement in an If

- What if you want to execute > 1 stmt in If?



## > 1 Stmt in an If

Does this work?

```
Statement1;
If (LogicalExpr)
  Statement2;
  Statement3;
  Statement4;
Statement5;
```

P

No, the indenting is irrelevant, section P is:

```
If (LogicalExpr)
  Statement2;
  Statement3;
  Statement4;
```

## > 1 Stmt in an If (A solution)

Solution - use a compound statement:

```
Statement1;
If (LogicalExpr) {
  Statement2;
  Statement3;
  Statement4;
}
Statement5;
```

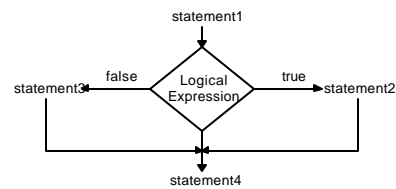
## Statement in If

Any statement reasonable as the single statement in if

- expression statement
  - assignment statement
  - function call (printf, scanf, etc.)
- compound statement
- if statement

## Two-Way Selection

- Sometimes desirable for statement to be executed when Logical Expression false

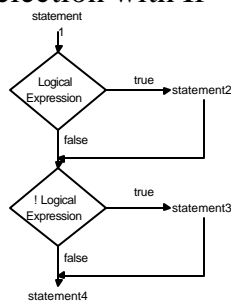


## Two-Way Selection with If

- Possible to use combinations of if:

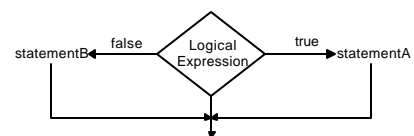
```
Statement1;
if (LogicalExpr)
  Statement2;
if (!(LogicalExpr))
  Statement3;
Statement4;
```

- But not efficient



## If-Else Statement

- Syntax:
- ```
if (LogicalExpr)
  StatementA
else
  StatementB
```



## FindMax with If-Else

```
int findMax(int Num1, int Num2) {
    int Big;

    Big = Num1;
    if (Num2 > Num1) Big = Num2; /* 1 test + */
    return Big; /* maybe 1 assign */
} /* test plus 1 or 2 assigns */

int findMax(int Num1, int Num2) {
    int Big;

    if (Num2 > Num1) /* 1 test + */
        Big = Num2; /* 1 assign or */
    else
        Big = Num1; /* 1 assign */
    return Big;
} /* test plus 1 assign */
```

## Using If-Else for Robustness

```
float calcAverage(float sum,
    int count) {
    if (count == 0)
        return 0.0;
    else
        return sum / count;
}
```

- Note return statement for each condition

## Compound Statements and If-Else

```
if ((year % 4) == 0) {
    printf("Leap year\n");
    numDays = 366;
}
else {
    printf("Not a leap year\n");
    numDays = 365;
}
```

## Programming Tip: Compound Stmt

- Does not hurt to always use compound statements for if, if-else statements
- ```
if (LogicalExpr) {
    /* statement or statements */
}

if (LogicalExpr) {
    /* statement(s) */
}
else {
    /* statement(s) */
}
```
- Easy to add statements later

## Programming Tip: Indenting

- Use indenting to make code more clear
  - indent statements in function definition to clearly identify body of function
  - indent statement(s) executed for if by fixed amount (2,3 chars) every time
  - for ifs within an if indent further
  - indent else statements(s) similarly
  - may want to make {,} match (on separate lines)

## Programming Tip: Conditions

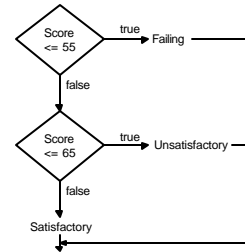
- Code most likely conditions first
- Code positive conditions where possible
- Code parts of solution executed most often first

## Multiway Selection

- Multiway if more than 2 alternatives
- Example:

Student Score	Message
0-55	Failing
56-65	Unsatisfactory
66-100	Satisfactory
- If-Else has two alternatives, to do multiway, string together If-Else statements

## Multiway Flow Chart



## Multiway with If-Else

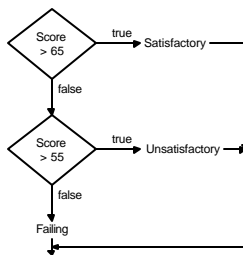
```
if (score <= 55)
    printf("Failing\n");
else
    if (score <= 65)
        printf("Unsatisfactory\n");
    else
        printf("Satisfactory\n");
```

## Indenting If-Else Combinations

- Multiway if-else statements sometimes indented:

```
if (score <= 55)
    printf("Failing\n");
else if (score <= 65)
    printf("Unsatisfactory\n");
else
    printf("Satisfactory\n");
```
- Rule for else: else matches most recent if

## Conditions Checked in Reverse



## Ordering Conditions

- ```
if (score > 65)
    printf("Satisfactory\n");
else if (score > 55)
    printf("Unsatisfactory\n");
else
    printf("Failing\n");
```
- But must check conditions in correct order
  - ```
if (score > 55)
    printf("Unsatisfactory\n");
else if (score > 65)
    printf("Satisfactory\n");
else
    printf("Failing\n");
```
  - Score of 70 would produce "Unsatisfactory"

## Multiway with If Statements

- Possible but inefficient:

```
if (score <= 55)
    printf("Failing\n");
if ((score > 55) && (score <= 65))
    printf("Unsatisfactory\n");
if (score > 65)
    printf("Satisfactory\n");
```

## Program Robustness

- Example assumes score in interval [0,100] but doesn't check

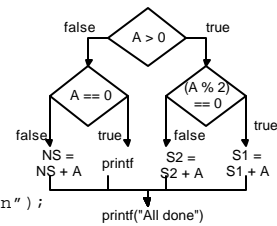
```
• Add test:
if ((score >= 0) && (score <= 100))
    /* print message */
else
    printf("Bad score: %d\n",score);
```

## Completed Code

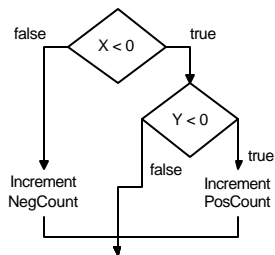
```
if ((score >= 0) && (score <= 100)) {
    if (score <= 55)
        printf("Failing\n");
    else if (score <= 65)
        printf("Unsatisfactory\n");
    else
        printf("Satisfactory\n");
}
else
    printf("Bad score: %d\n",score);
```

## Nesting Example

```
if (A > 0)
    if ((A % 2) == 0)
        S1 = S1 + A;
    else
        S2 = S2 + A;
else
    if (A == 0)
        printf("A zero\n");
    else
        NS = NS + A;
    printf("All done.\n");
```



## Matching Else Example



## Matching Single Else

- Easy to make mistake

```
if (X < 0)
    if (Y < 0)
        PosCount++;
else
    NegCount++;
```

- Despite indenting, else matches wrong if
- PosCount updated when X < 0, Y >= 0

## Matching Single Else Solutions

```

if (X < 0)
  if (Y < 0)
    PosCount++;
  else
    ; /* empty */
else
  NegCount++;

if (X < 0) {
  if (Y < 0)
    PosCount++;
} /* compound stmt */
else
  NegCount++;

if (X >= 0)
  NegCount++;
else
  if (Y < 0)
    PosCount++;
  
```

## Choosing Conditions

Selection	Example	C Statement
One way	$X > 0$	If
Two way	$X > 0$ $X \leq 0$	If-Else
Multiway	$0 \leq X \leq 10$ $11 \leq X \leq 20$ $21 \leq X \leq 30$	Nested If-Elses

## Another Multiway Method

• Consider another grading problem:

Score	Grade
9-10	A
7-8	B
5-6	C
0-4	F

```

if ((score == 9) ||
    (score == 10))
  grade = 'A';
else if ((score == 7) ||
         (score == 8))
  grade = 'B';
else if ((score == 5) ||
         (score == 6))
  grade = 'C';
else
  grade = 'F';
  
```

## Switch Statement

• More readable approach: switch statement

```

switch (score) {
  case 9: case 10:
    grade = 'A';
    break;
  case 7: case 8:
    grade = 'B';
    break;
  case 5: case 6:
    grade = 'C';
    break;
  case 0: case 1: case 2: case 3: case 4:
    grade = 'F';
}
  
```

## Switch Format

```

switch (Expression) {
  case const1-1: case const1-2: ...
    statement
    statement
  ...
  case const2-1: case const2-2: ...
    statement
    statement
  ...
  default: ...
    statement
    statement
  ...
}
  
```

## Switch Rules

- Expression must be integral type (no float)
- Case labels must be constant values
- No two case labels with same value
- Two case labels can be associated with same set of statements
- Default label is not required
- At most one default label

## Switch Rules

### Evaluating

- determine value of expression
- look for corresponding label
- if no corresponding label look for default
- if no corresponding label or default, do nothing
- execute all statements from label to }

ALL statements from label to } ???

## Executing Switch Example

```
switch (score) {
  case 9: case 10:
    grade = 'A';
  case 7: case 8:
    grade = 'B';
  case 5: case 6:
    grade = 'C';
}
```

- score is 7:  
grade = 'B'
- score is 5:  
grade = 'C'
- score is 9:  
grade = 'A'
- not quite what we want

## The break Statement

- break used to indicate a set of statements is finished (and no more should be executed)
- syntax: `break;`
- break says to stop executing and go to the next } (skipping any statements in between)
- add after each set of cases

## default case

The default case can be used to deal with robustness issues (score is < 0 or > 10)

```
switch (score) {
  case 9: case 10:
    grade = 'A';
    break;
  case 7: case 8:
    grade = 'B';
    break;
  case 5: case 6:
    grade = 'C';
    break;
  case 0: case 1: case 2: case 3: case 4:
    grade = 'F';
    break;
  default:
    printf("Bad score %d\n",score);
}
```

## Other Expression Types

- Any integral type can be used as expression
- Cases must match

```
char married;
switch (married) {
  case 'S': case 's':
    printf("single"); break;
  case 'D': case 'd':
    printf("divorced"); break;
  case 'M': case 'm':
    printf("married");
}
```

## When Not to Use Switch

- Example: Case  $0 \leq X \leq 100$ 

```
switch (x) {
  case 0: case 1: case 2: case 3:
  case 4: case 5: case 6: case 7:
    ...
  case 100:
    ...
}
```
- Better to use nested ifs