

Selection Structures: if and switch Statements

SDC OSW 3541

**Department of Computer Science & Engineering
ITER, Siksha 'O' Anusandhan Deemed To Be University
Jagamohan Nagar, Jagamara, Bhubaneswar, Odisha - 751030**

Text Book(s)



Jeri R Hanly, & Elliot B. Koffman

Problem Solving & Program Design in C

Seventh Edition

Pearson Education



Robert Love

Linux System Programming

Second Edition, ISBN: 978-1-449-33953-1

O'REILLY

Talk Flow

- 1 Introduction
- 2 Control Structures
- 3 if Statement
- 4 Nested if Statements and Multiple-Alternative Decisions
- 5 The switch Statement
- 6 Common Programming Errors
- 7 Practice Questions

Introduction

- This chapter begins your study of statements that control the flow of the program execution.
- You will learn to use if and switch statements to select one statement group to execute from many alternatives.
- The case studies in this chapter emphasize reusing solutions to prior problems to speed up the problem-solving process.
- You will also learn how to trace an algorithm or program to verify that it does what you expect.

Control Structures

- **Control structures** control the flow of execution in a program or function.
- Instructions are organized into three control structures to control execution flow: sequence, selection, and repetition.
- A compound statement, written as a group of statements bracketed by { and }, is used to specify sequential flow.

```
{  
    statement 1;  
    statement 2;  
    .  
    .  
    .  
    statement n;  
}
```

Conditions

- A program chooses among alternative statements by testing the value of key variables.
- For example, one indicator of the health of a person's heart is the resting heart rate. A program that gets a person's resting heart rate as data should compare that value to 75 and display a warning message if the rate is over 75.
- If *rest_heart_rate* is a type int variable, the expression will be

rest_heart_rate > 75

- Most conditions that we use to perform comparisons will have one of these forms:

variable	relational-operator	variable
variable	relational-operator	constant
variable	equality-operator	variable
variable	equality-operator	constant

Relational and Equality Operators

Table 1: Relational and Equality Operators

Operator	Meaning	Type
<	less than	relational
>	greater than	relational
<=	less than or equal to	relational
>=	greater than or equal to	relational
==	equal to	equality
!=	not equal to	equality

x	power	MAX_POWER	y	item	MIN_ITEM	mom_or_dad	num	SENTINEL
-5	1024	1024	7	1.5	-999.0	'M'	999	999

Operator	Condition	English Meaning	Value
<=	x <= 0	x less than or equal to 0	1 (true)
<	power < MAX_POW	power less than MAX_POW	0 (false)
>=	x >= y	x greater than or equal to y	0 (false)
>	item > MIN_ITEM	item greater than MIN_ITEM	1 (true)
==	mom_or_dad == 'M'	mom_or_dad equal to 'M'	1 (true)
!=	num != SENTINEL	num not equal to SENTINEL	0 (false)

Logical Operators

With the three logical operators `&&` *and*, `||` *or*, `!` *not* —we can form more complicated conditions or logical expressions.

Example of logical expression or (`||`)

```
salary < MIN_SALARY || dependents > 5
```

The logical expression evaluates to 1 (true) if either the condition

```
salary < MIN_SALARY
```

or the condition

```
dependents > 5
```

is true.

Relational and Equality Operators (contd.)

Example of logical expression and(&&)

```
temperature > 90.0 && humidity > 0.90
```

The logical expression evaluates to 1 (true) if both the condition

```
temperature > 90.0
```

and the condition

```
humidity > 0.90
```

are true.

Example of logical expression not(!)

```
!(0 <= n && n <= 100)
```

The logical expression evaluates to 1 (true) if the inner condition

```
0 <= n && n <= 100
```

is 0 (false).

Relational and Equality Operators (contd.)

Table 2: The && Operator (and)

operand1	operand2	operand1 && operand2
nonzero (true)	nonzero (true)	1 (true)
nonzero (true)	0 (false)	0 (false)
0 (false)	nonzero (true)	0 (false)
0 (false)	0 (false)	0 (false)

Table 3: The || Operator (or)

operand1	operand2	operand1 operand2
nonzero (true)	nonzero (true)	1 (true)
nonzero (true)	0 (false)	1 (true)
0 (false)	nonzero (true)	1 (true)
0 (false)	0 (false)	0 (false)


Table 4: The ! Operator (not)

operand1	!operand1
nonzero (true)	0 (false)
0 (false)	1 (true)

Operator Precedence

An operator's precedence determines its order of evaluation depicted in following table lists the precedence of all C operators so far, from highest to lowest.

Operator Precedence

Operator	Precedence
function calls	highest
! + - & (unary operators)	
* / %	
+ -	
< <= >= >	
== !=	
&&	
=	
	lowest

Relational and Equality Operators (contd.)

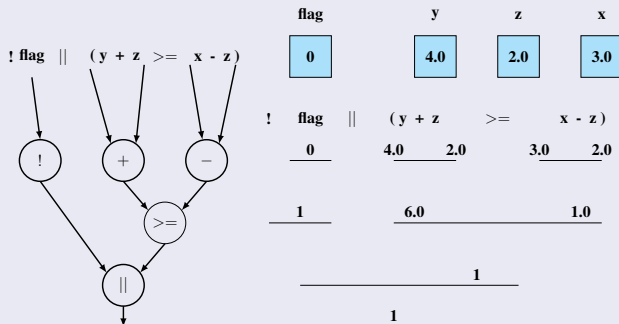
Operator Precedence Example

Lets consider the example :

!flag || (y + z >= x - z)

where,

x	y	z	flag
3.0	4.0	2.0	0



Short-Circuit Evaluation

- Technique of stopping evaluation of a logical expression as soon as its value can be determined is called **short-circuit** evaluation.
- An expression of the form `a || b` must be true if `a` is true. Consequently, C stops evaluating the expression `b`.
- An expression of the form `a && b` must be false if `a` is false and hence `b` is not evaluated.
- Lets consider the example :

`!flag || (y + z >= x - z)`

x y z flag

3.0	4.0	2.0	0
-----	-----	-----	---

where,

- C stops evaluating the expression when it determines that the value of `!flag` is 1 (true).

Short-Circuit Evaluation

- **Advantages Of Short-Circuit Evaluation:**

- It can be helpful in avoiding computationally expensive tasks under certain circumstances.
- It provides a check for the first argument without which the second argument may result in a runtime error.

- **Disadvantages Of Short-Circuit Evaluation:**

- It can cause unexpected behavior if not used properly. If some operation that does some kind of allocation of system resources/memory allocation gets skipped due to short-circuiting, we may get unexpected behavior.
- Code execution becomes less efficient with short-circuited execution paths because in some compilers the new checks for short-circuits are extra execution cycles in themselves.

Relational and Equality Operators (contd.)

DeMorgan's Theorem

- The complement of $\text{expr 1} \ \&\& \ \text{expr 2}$ is written as $\text{comp 1} \ || \ \text{comp 2}$, where comp 1 is the complement of expr 1 , and comp 2 is the complement of expr 2 .
- The complement of $\text{expr 1} \ || \ \text{expr 2}$ is written as $\text{comp 1} \ \&\& \ \text{comp 2}$, where comp 1 is the complement of expr 1 , and comp 2 is the complement of expr 2 .

Example (try yourself)

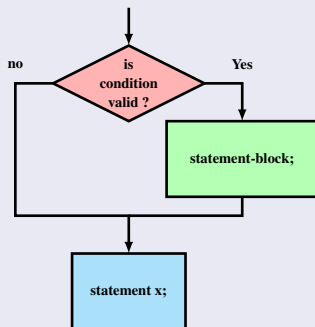
1. Write an expression to test for each of the following relationships.
 - a. age is from 18 to 21 inclusive.
 - b. water is less than 1.5 and also greater than 0.1 .
 - c. year is divisible by 4 . (Hint: Use % .)
 - d. speed is not greater than 55 .
 - e. y is greater than x and less than z .
 - f. w is either equal to 6 or not greater than 3 .

if Statement

Simple if Statement - Structure

The general form of a simple if statement is:

```
if (test_condition)
{
    statement-block;
}
statement x;
```



if Statement (contd.)

Example

```
/*Print if password 1234 is valid*/
#include<stdio.h>
int main()
{
    int a;
    printf("Enter a password : ");
    scanf("%d", &a);
    if(a == 1234)
    {
        printf("You are password is valid\n",a);
    }
    return 0;
}
```

Output

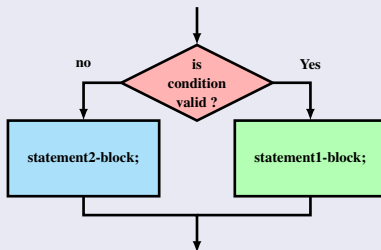
```
Enter a password : 1234
You are password is valid
```

if Statement (contd.)

if Statement with Two Alternatives

if...else statement : An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

```
if(boolean_expression) {  
    statement1-block;  
}  
else{  
    statement2-block;  
}  
statement x;
```



if Statement (contd.)

Example

```
/*Find a number entered by the user is even or odd*/
#include<stdio.h>
int main()
{
    int a;
    printf("Enter a number : ");
    scanf("%d", &a);
    if(a%2 == 0)
    {
        printf("%d is even\n", a);
    }
    else
    {
        printf("%d is odd\n");
    }
    return 0;
}
```

Output

```
Enter a number : 13
13 is odd
```

if Statement(contd.)

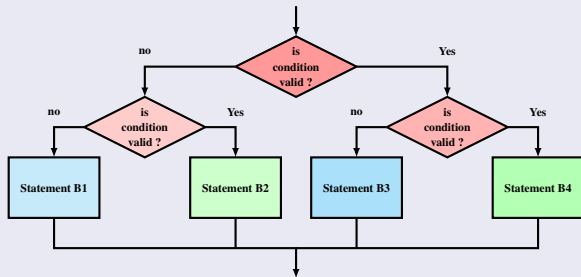
Example (try yourself)

1. Write C statements to carry out the following steps.
 - a. If item is nonzero, then multiply product by item and save the result in product ; otherwise, skip the multiplication. In either case, print the value of product .
 - b. Store the absolute difference of x and y in y, where the absolute difference is $(x - y)$ or $(y - x)$, whichever is positive. Do not use the abs or fabs function in your solution.
 - c. If x is 0 , add 1 to zero_count . If x is negative, add x to minus_sum . If x is greater than 0 , add x to plus_sum .

Nested if Statements

Nested if statement an if statement with another if statement as its true task or its false task.

```
if(boolean_expression 1)
{
    if( boolean_expression 2)
    {
        Statement-Block 1;
    }
    else
    {
        Statement-Block 2;
    }
}
else
{
    if( boolean_expression 2)
    {
        Statement-Block 3;
    }
    else
    {
        Statement-Block 4;
    }
}
```



Nested if Statements (contd.)

Program to find maximum integer among 3 input

```
#include<stdio.h>
int main(void){
    int a,b,c, max;
    printf("Enter the 3 numbers :");
    scanf("%d %d %d", &a, &b, &c);
    if(a>=b){
        if(a>=c){
            max = a;
        }
        else
        {
            max=c;
        }
    }
    else{
        if(b>=c)
        {
            max = b;
        }
        else
        {
            max=c;
        }
    }
    printf("%d is max among %d, %d, and %d\n", max, a, b, c);
}
```

if Statement(contd.)

Output

Sample Run 1

Enter the 3 numbers : 2 3 1

3 is max among 2, 3, and 1

Sample Run 2

Enter the 3 numbers : 12 3 18

18 is max among 12, 3, and 18

Multiple-Alternative Decisions

- Multiple-Alternative Decision.

```
if(boolean_expression 1)
{
    /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2)
{
    /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3)
{
    /* Executes when the boolean expression 3 is true */
}
else
{
    /* executes when the none of the above condition is true */
}
```


Multiple-Alternative Decisions (contd.)

```
//Example
//Assign student the grade according to its total mark
//Mark   >   90           :   'O'
//Mark   >   80  <=   90   :   'A'
//Mark   >   70  <=   80   :   'B'
//Mark   >   60  <=   70   :   'C'
//Mark   >   50  <=   60   :   'D'
//Mark           <   50   :   'F'
#include<stdio.h>
int main(){
    float mark; char grade;
    printf("enter the total mark obtained by student (max 100): ");
    scanf("%f", &mark);
    if(mark > 100){
        printf("wrong mark ....\n");
        return 0;}
    else if(mark>90)
        grade = 'O';
    else if(mark>80)
        grade = 'A';
    else if(mark>70)
        grade = 'B';
    else if(mark>60)
        grade = 'C';
    else if(mark>50)
        grade = 'D';
    else
        grade = 'F';
    printf("The student has obtained %c grade in exam\n", grade);
    return 0;}
```

Multiple-Alternative Decisions (contd.)

Output 1:

```
enter the total mark obtained by student (max 100): 95
The student has obtained O grade in exam
```

Output 2:

```
enter the total mark obtained by student (max 100): 55
The student has obtained E grade in exam
```

Output 3:

```
enter the total mark obtained by student (max 100): 120
wrong mark ....
```

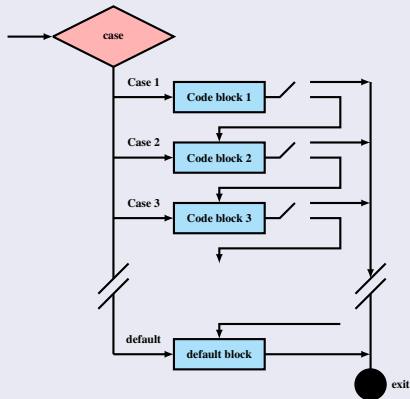
switch Statement

switch Statement

- The complexity of a program increases dramatically when the number of control statements increases.
- Fortunately, C has a built-in multiway decision-making statement known as switch.

SYNTAX:

```
switch ( controlling expression ) {  
    label set_1:  
    statements_1  
    break;  
    label set_2:  
    statements_2  
    break;  
    ...  
    ...  
    label set_n:  
    statements_n  
    break;  
    default:  
    statements_d  
}
```



Program Using a switch Statement for Selection

Reads serial number and displays class of ship

```
#include <stdio.h>
int main(void){
    char class;          /* input - character indicating class of ship */
    /* Read first character of serial number */
    printf("Enter ship serial number> ");
    scanf("%c", &class); /* scan first letter */
    /* Display first character followed by ship class */
    printf("Ship class is %c: ", class);
    switch (class) {
        case 'B':
        case 'b':
            printf("Battleship\n");
            break;
        case 'C':
        case 'c':
            printf("Cruiser\n");
            break;
        case 'D':
        case 'd':
            printf("Destroyer\n");
            break;
        case 'F':
        case 'f':
            printf("Frigate\n");
            break;
        default:
            printf("Unknown\n");
    }
    return (0);}
```

Common Programming Errors

Programming Errors

The fact that C relational and equality operators give a result of 1 for true and 0 for false means that C interprets some common mathematical expressions in a way that seems surprising at first. You would probably not anticipate the fact that the following if statement displays Condition is true for all values of x.

```
if (0 <= x <= 4)
    printf("Condition is true\n");
```

For example, let's consider the case when x is 5 . The value of $0 \leq 5$ is 1 , and 1 is certainly less than or equal to 4! In order to check if x is in the range 0 to 4 , you should use the condition

Programming Errors

```
(0 <= x && x <= 4)
```

Remember that the C equality operator is `==` . If you slip up and use `=` , the mathematical equal sign, the compiler can detect this error only if the first operand is not a variable. Otherwise, your code will simply produce incorrect results. For example, the code fragment that follows always prints `x is 10`, regardless of the value of `x` .

```
if (x = 10)
    printf("x is 10");
```

The assignment operator stores the value 10 in `x` . The value of an assignment expression is the value assigned, so in this case the value of the `if` condition of the statement is 10 . Since 10 is nonzero, C views it as meaning true and executes the true task.

Practice Questions

Q1

Write a program that calculates the user's body mass index (BMI) and categorizes it as underweight, normal, overweight, or obese, based on the following table from the United States Centers for Disease Control:

BMI	Weight Status
Below 18.5	Underweight
18.5–24.9	Normal
25.0–29.9	Overweight
30.0 and above	Obese

To calculate BMI based on weight in pounds (*wt_lb*) and height in inches (*ht_in*), use this formula (rounded to tenths):

$$\frac{703 \times wt_lb}{ht_in^2}$$

Prompt the user to enter weight in pounds and height in inches.

Practice Questions (contd.)

Q2

While spending the summer as a surveyor's assistant, you decide to write a program that transforms compass headings in degrees (0 to 360) to compass bearings. A compass bearing consists of three items: the direction you face (north or south), an angle between 0 and 90 degrees, and the direction you turn before walking (east or west). For example, to get the bearing for a compass heading of 110.0 degrees, you would first face due south (180 degrees) and then turn **70.0** degrees east ($180.0 - 70.0 = 110.0$). Therefore, the bearing is South **70.0** degrees East. Be sure to check the input for invalid compass headings.

Practice Questions (contd.)

Q3

Write a program that reports the contents of a compressed-gas cylinder based on the first letter of the cylinder's color. The program input is a character representing the observed color of the cylinder: 'Y' or 'y' for yellow, 'O' or 'o' for orange, and so on. Cylinder colors and associated contents are as follows:

orange	ammonia
brown	carbon monoxide
yellow	hydrogen
green	oxygen

Your program should respond to input of a letter other than the first letters of the given colors with the message, **Contents unknown** .

Practice Questions (contd.)

Q4

The National Earthquake Information Center has asked you to write a program implementing the following decision table to characterize an earthquake based on its Richter scale number.

Richter Scale Number (n)	Characterization
$n < 5.0$	Little or no damage
$5.0 \geq n < 5.5$	Some damage
$5.5 \geq n < 6.5$	Serious damage: walls may crack or fall
$6.5 \geq n < 7.5$	Disaster: houses and buildings may collapse
higher	Catastrophe: most buildings destroyed

Could you handle this problem with a switch statement? If so, use a switch statement; if not, explain why.

Practice Questions (contd.)

Q5

Write a program that determines the day number (1 to 366) in a year for a date that is provided as input data. As an example, January 1, 1994, is day 1. December 31, 1993, is day 365. December 31, 1996, is day 366, since 1996 is a leap year. A year is a leap year if it is divisible by four, except that any year divisible by 100 is a leap year only if it is divisible by 400. Your program should accept the month, day, and year as integers. Include a function `leap` that returns 1 if called with a leap year, 0 otherwise.

Practice Questions (contd.)

Q6

Write a program to control a bread machine. Allow the user to input the type of bread as **W** for White and **S** for Sweet. Ask the user if the loaf size is double and if the baking is manual. The following table details the time chart for the machine for each bread type. Display a statement for each step. If the loaf size is double, increase the baking time by 50 percent. If baking is manual, stop after the loaf-shaping cycle and instruct the user to remove the dough for manual baking. Use functions to display instructions to the user and to compute the baking time.

BREAD TIME CHART

Operation	White Bread	Sweet Bread
Primary kneading	15 mins	20 mins
Primary rising	60 mins	60 mins
Secondary kneading	18 mins	33 mins
Secondary rising	20 mins	30 mins
Loaf shaping	2 seconds	2 seconds
Final rising	75 mins	75 mins
Baking	45 mins	35 mins
Cooling	30 mins	30 mins