C Program Control – Selection

Deitel 8th Edition, Chapter 4

Topics

Control Structures Review

Conditions

Logical Expressions

The if Statement

The switch Statement

Integral Data

Confusing the equality and assignment operators

fflush

CONTROL STRUCTURES REVIEW

What is a Control Structure?

Statement(s) that combine individual instructions into:

- a single logical unit
- with one entry point and
- one exit point

“Controls” the flow of execution

THREE KINDS OF CONTROL STRUCTURES

Compound statement / sequential control

Selection control structure

Repetition control structure

Conditions

Selection Control Structure

Selection means there is a choice

Selection control structure provides different execution paths through a section of code

Which statement to execute is determined by checking the value of some variable using a **condition**

Condition can also be called a condition statement or a **conditional expression**

What is a Condition?

A condition is an **expression**

An expression has a **value**

The value of a condition is **either false (0) or true (non-zero)**

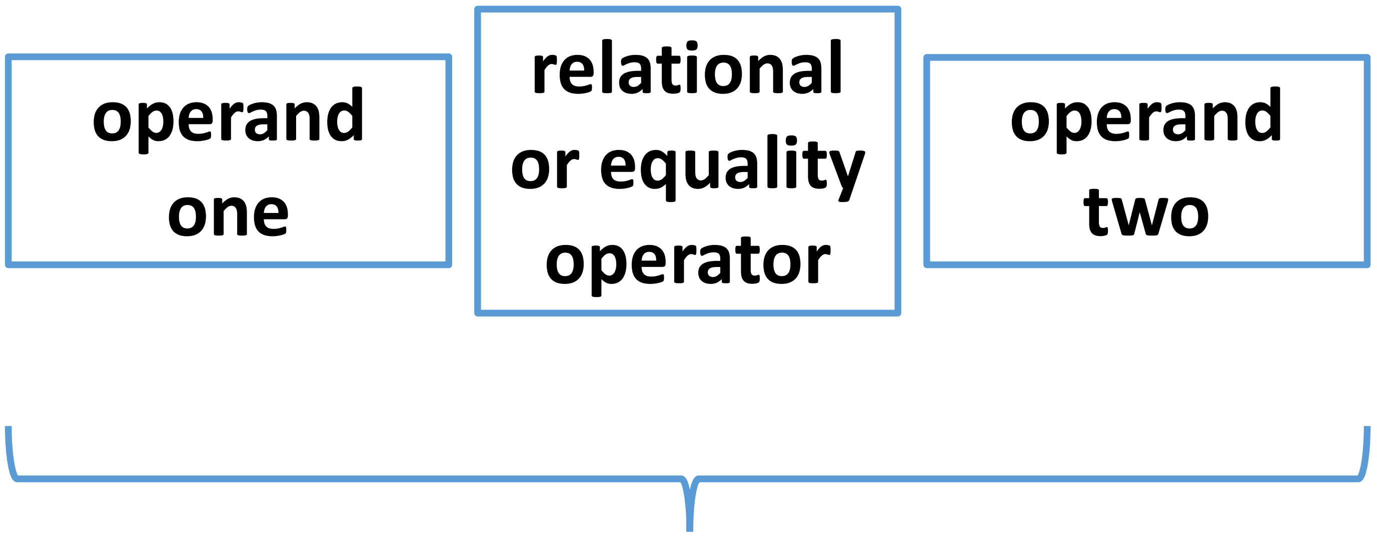
This value sets the criterion for executing or skipping a statement or group of statements

Conditional expressions use relational and equality operators

Relational and Equality Operators in C

Operator	Meaning	Type of Operator
<	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

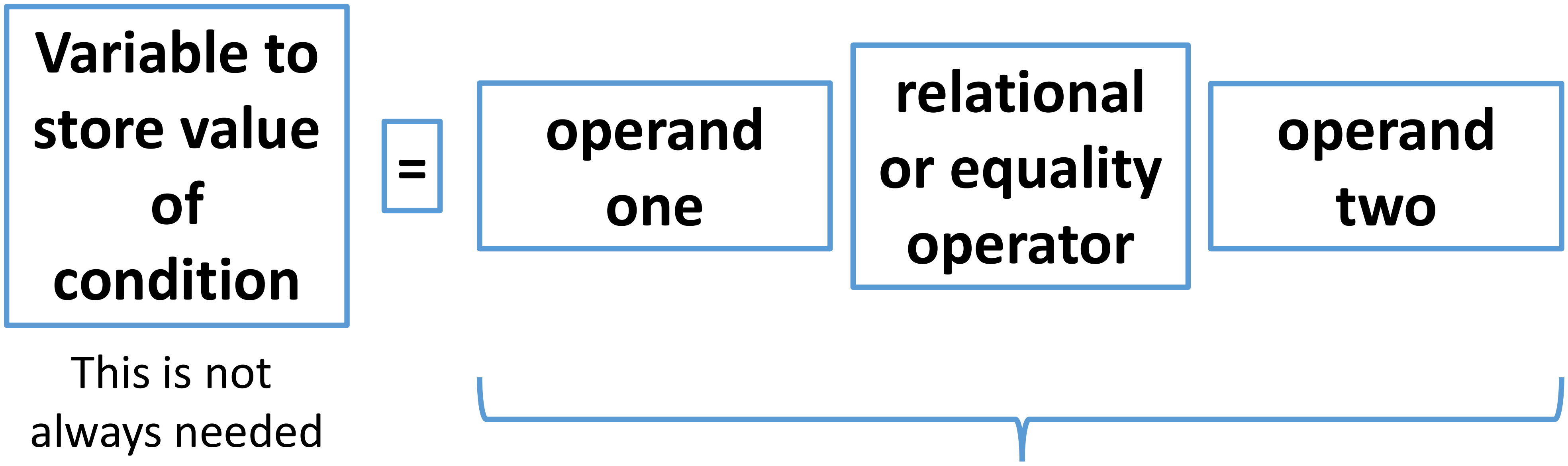
General Format of a Condition, aka Conditional Expression



This is a **conditional expression**, which has a value

CONDITIONAL EXPRESSION IN AN ASSIGNMENT STATEMENT

If we want to keep the value for use later, it needs to be assigned to a variable


Variable to store value of condition
=
operand one
relational or equality operator
operand two

This is not always needed

This is a conditional expression, which has a value

LOGICAL EXPRESSIONS

We Can Combine Two or More Conditions

This is called a **logical expression**

$$\boxed{\textbf{Variable to store value of logical condition}} \; \boxed{=} \; \underbrace{\boxed{\textbf{condition one}}\;\boxed{\textbf{logical operator}}\;\boxed{\textbf{condition two}}}$$

This is not always needed

This is a logical expression, which has a value

Logical Operators

A logical expression uses one or more **logical operators**

Can build complex expressions using more than one simple conditional expression

Logical operators:

Operator	Meaning
&&	and
||	or
!	not; negation

Logical Expression Values

Operator	Logical Expression is TRUE if:	Logical Expression is FALSE if:
&& (and)	Both conditions are true	Either condition is false
|| (or)	Either condition is true	Both condition are false
! (not)	Condition is false	Condition is true

&& (AND) LOGICAL EXPRESSION TRUTH TABLE

Value of condition 1	Value of condition 2	Value of cond1 && cond2
true	true	true
true	false	false
false	true	false
false	false	false

| | (OR) Logical Expression Truth Table

Value of condition 1	Value of condition 2	Value of cond1 || cond2
true	true	true
true	false	true
false	true	true
false	false	false

! (NOT) LOGICAL EXPRESSION TRUTH TABLE

Value of condition	**Value of ! (cond1)**
true	false
false	true

Short-Circuit Evaluation

Sometimes, we don’t need to perform a **complete** evaluation of an expression, if the only possible value of the expression becomes obvious before it is completely evaluated

Short-Circuit Evaluation Example 1

condition1 || condition2

If condition1 is true, the expression will be true, so there is no need to evaluate condition2

If condition1 is false, we don’t yet know if the expression is true or false, so condition2 must be evaluated

Short-Circuit Evaluation Example 2

condition1 && condition2

If condition1 is false, the expression will be false, so there is no need to evaluate condition2

If condition1 is true, we don't yet know if the expression is true or false, so condition2 must be evaluated

| | (OR) LOGICAL EXPRESSION TRUTH TABLE – SHORT CIRCUIT EVALUATION 1

Value of condition 1	Logical expression	Does short circuit evaluation happen?
true	true || *condition2*	
false	false || *condition2*	

| | (OR) LOGICAL EXPRESSION TRUTH TABLE – SHORT CIRCUIT EVALUATION 2

Value of condition 1	**Logical expression**	**Does short circuit evaluation happen?**
true	true || *condition2*	**Yes.** Condition 2 does not need to be evaluated.
false	false || *condition2*	**No.** Condition 2 needs to be evaluated.

&& (AND) LOGICAL EXPRESSION TRUTH TABLE – SHORT CIRCUIT EVALUATION 1

Value of condition 1	**Logical expression**	**Does short circuit evaluation happen?**
true	true && *condition2*	
false	false && *condition2*	

&& (AND) LOGICAL EXPRESSION TRUTH TABLE – SHORT CIRCUIT EVALUATION 2

Value of condition 1	**Logical expression**	**Does short circuit evaluation happen?**
true	true && *condition2*	**No.** Condition 2 needs to be evaluated.
false	false && *condition2*	**Yes.** Condition 2 does not need to be evaluated.

Operator Precedence

Order	Operator Type	Operator
1	Parentheses	( )
2	Function calls	
3	Unary operators	! + — &
4	Binary operators (MDR)	* / %
5	Binary operators (AS)	+ —
6	Relational operators	< <= > >=
7	Equality operators	== !=
8	Logical operators	&& ||
9	Assignment operator	=

Comparing Characters

We can compare characters in C using the character’s ASCII code

Are these true or false?

'9' >= '1'

'B' <= 'A'

‘@' >= 'Z'

'A' <= 'a'

Comparing Characters Results

We can compare characters in C using the character’s ASCII code

Are these true or false?

'9' >= ‘1’	ASCII codes: ‘9’ is 57, ‘1’ is 49	This is true
'B' <= ‘A’	ASCII codes: ‘B’ is 66, ‘A’ is 65	This is false
‘@' >= ‘Z’	ASCII codes: ‘@’ is 64, ‘Z’ is 90	This is false
'A' <= 'a'	ASCII codes: ‘A’ is 65, ‘a’ is 97	This is true

Logical Assignment

Simplest logical expression uses an integer, or a variable that is meant to represent true or false

Use the standard assignment statement to set these variables to true (a non-zero value) or false (0)

Don’t use doubles or floats to represent true – why not?

Selection Control in C – The if Statement

Types of if Statements

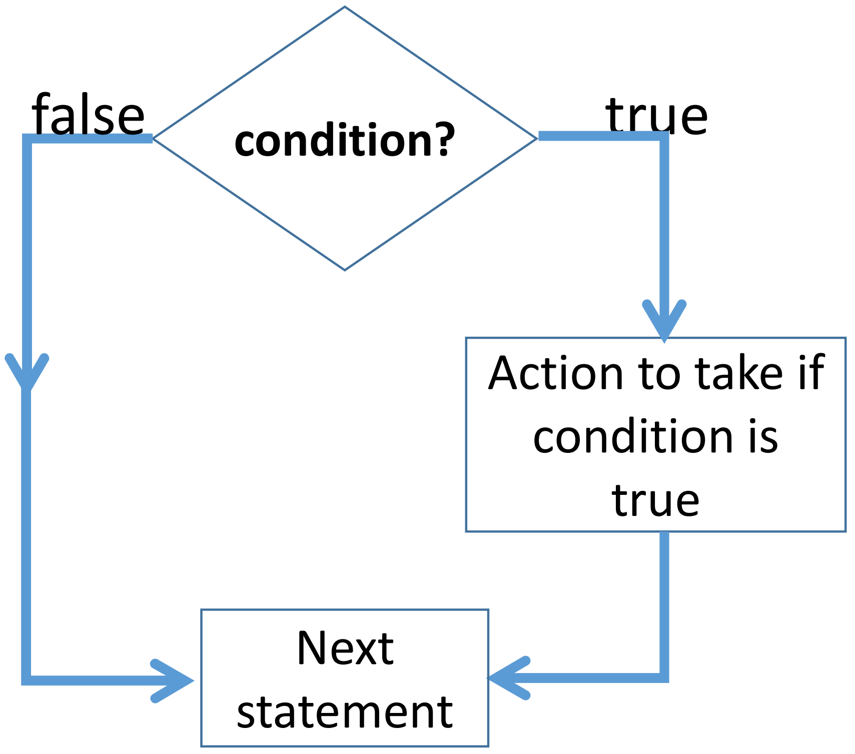
Selection control in C can use the **if** statement

Three types of **if** statements:

1. One alternative or possible action
2. Two alternatives or possible actions
3. Multiple alternatives

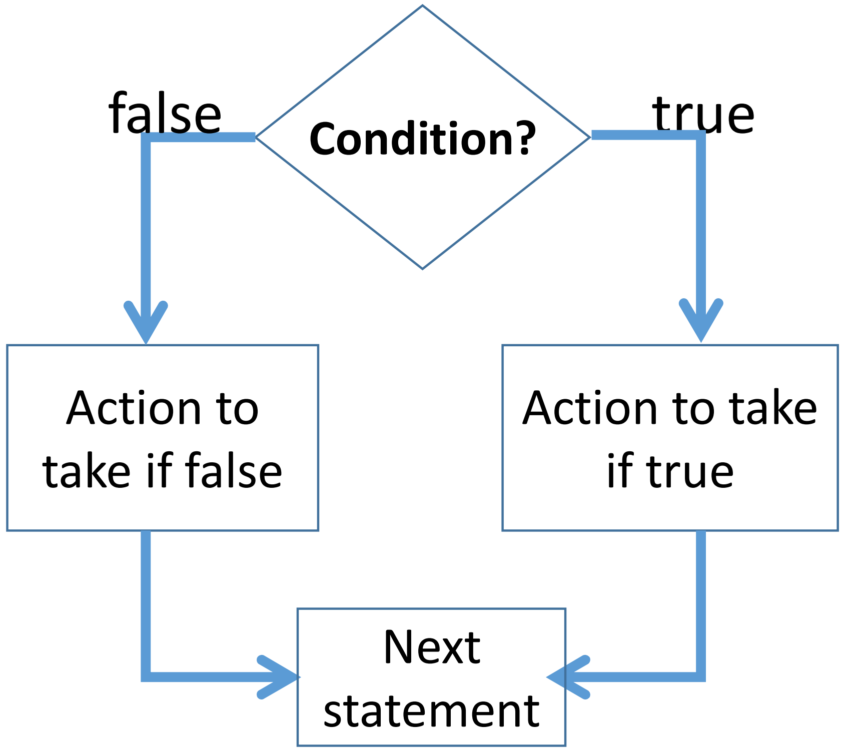
General Format – One Alternative

```
if (condition expression)
    action to take if expression true;
```



General Format – Two Alternatives

```
if (condition expression)
    action to take if expression is true;
else
    action to take if expression is false;
```



Multiple Actions within if

Usually more than one action to perform if a condition is true or false.

```
if (condition expression)

{
     action1 to take if expression is true;
     action2 to take if expression is true;
     action3 to take if expression is true;
}

else

    action to take if expression is false;
```

MULTIPLE ACTIONS WITHIN IF – WHAT IS MORE THAN ONE STATEMENT?

Usually more than one action to perform if a condition is true or false.

```
if (condition expression)

{
     action1 to take if expression is true;
     action2 to take if expression is true;
     action3 to take if expression is true;
}

else

    action to take if expression is false;
```

$\left.\vphantom{\begin{array}{c}\{\\a\\a\\a\\\}\end{array}}\right\}$ What kind of control structure is this part?

Multiple Actions within else

Usually more than one action to perform if a condition is true or false.

```
if (condition expression)

     action to take if expression is true;

else
{
     action1 to take if expression is false;
     action2 to take if expression is false;
     action3 to take if expression is false;
}
```

IF STATEMENTS WITH COMPOUND STATEMENTS EXAMPLE 1

Example – Sort two values, x and y, so that the smaller value is stored in x, and the larger value is stored in y.

What are the possible current conditions here?

1.

2.

3.

IF STATEMENTS WITH COMPOUND STATEMENTS EXAMPLE 1 CONDITIONS

What should we do in each situation?

1. x < y
2. x > y
3. x equals y

if Statements with Compound Statements Example 1 Actions

What should we do in each situation?

1. x < y	Nothing – they're in the correct order
2. x > y	Swap values of x and y
3. x equals y	Nothing

What kind of **if** statement is needed?

How many alternatives are there?

if Statements with Compound Statements Example 1 Situations

1. x < y	Nothing – they’re in the correct order
2. x > y	Swap values of x and y
3. x equals y	Nothing

There are three situations, but only one action to take (swapping)

But swapping the values of two variables requires three statements

We can use single selection, with a nested compound statement

Example: Nested if Statements

Read in a number, and determine if it is positive, negative, or neither (zero)

What are the possibilities for the number?

1.

2.

3.

Example: Nested if Statements Conditions

What should we do in each situation?

1. number > 0

2. number < 0

3. number equals 0

EXAMPLE: NESTED IF STATEMENTS ACTIONS

What should we do in each situation?

1. number > 0	print “Positive”
2. number < 0	print “Negative”
3. number equals 0	print “Zero”

What kind of **if** statement is needed?

How many alternatives are there?

Example: Nested if Statements Situations

What should we do in each situation?

1. number > 0	print "Positive"
2. number < 0	print "Negative"
3. number equals 0	print "Zero"

There are three situations, each with a different action to take.

We need multiple selection, with a nested if statement

NESTED IF STATEMENTS

Use nested if statements to handle multiple alternatives

```
if (condition expression)                          ← outer if structure start
    action to take if expression is true;
else
    if (condition expression 2)                    ← inner (nested) if structure start
        action to take if expression is true;
    else
        action to take if expression is false;     ← inner (nested) if structure end
//outer if structure end
```

Nested if Statements To Check for Positive, Negative, Zero

```
if (x > 0)
      printf("Positive");
else
      {
            if (x < 0)
                  printf("Negative");
            else
                  printf("Zero");
      }
```

Many Nested if Statements Can Become Unwieldy

Nesting many if statements like this can become complex and confusing to write and read

Another option is to use the multiple-alternative format of the if statement

IF STATEMENT: MULTIPLE-ALTERNATIVE FORMAT

```
if (condition expression 1)
    action to take if expression 1 is true;
else if (condition expression 2)
    action to take if expression 2 is true;
…
else if (condition expression n)
    action to take if expression n is true;
else
    action to take if expression n is false;
```

IF STATEMENT: MULTIPLE-ALTERNATIVE FORMAT

Rewrite of previous example:

```
if (x > 0)
    printf("Positive");
else if (x < 0)
    printf("Negative");
else
    printf("Zero");
```

Order of Conditions with Multiple Alternatives

Only the action following the **first true condition** is executed

Putting the conditions in the correct order is important!

SELECTION CONTROL IN C – THE SWITCH STATEMENT

The switch Statement

When we have multiple alternatives, we can also use a **switch statement**

Useful when the alternative actions are based on the **value of a single variable or a simple expression**

The variable or expression can be an **int** or a **char** only

Switch Statement – General Format

```
switch (single variable or simple expression)  {

  case constant_one:
         action(s) to take if controlling expression matches constant_one;
         break;
  …
  case constant_n:
         action(s) to take if controlling expression matches constant_one;
         break;

  default:
         action(s) to take if controlling expression matches no cases
  }
```

SWITCH STATEMENT DETAILS – SWITCH

switch (single variable or simple expression)

The part in () is called the **controlling expression**

Must evaluate to an **integral value** (more on this in a bit)

SWITCH STATEMENT DETAILS – CASES

```
case constant_one:

case constant_n:
```

Called cases

The keyword **case** is required

The **case label** follows case and must be an **integral constant**

Colon at end is required

SWITCH STATEMENT DETAILS – IF THERE IS A MATCH

```
case constant_one:
        action(s) to take if controlling expression matches constant_one;
        break;
```

If controlling expression matches a case label:

All statements below case are executed, until the **break statement** is encountered

SWITCH STATEMENT DETAILS – DEFAULT CASE

default:
        action(s) to take if controlling expression matches no cases

Default case – executed if no other case matches

Useful for error messages when there is no match

Not required if there is nothing to do when there is no match

Listed last

SWITCH STATEMENT DETAILS – CURLY BRACES

Curly braces **not used around individual cases**, but **are required** around the body of the switch itself

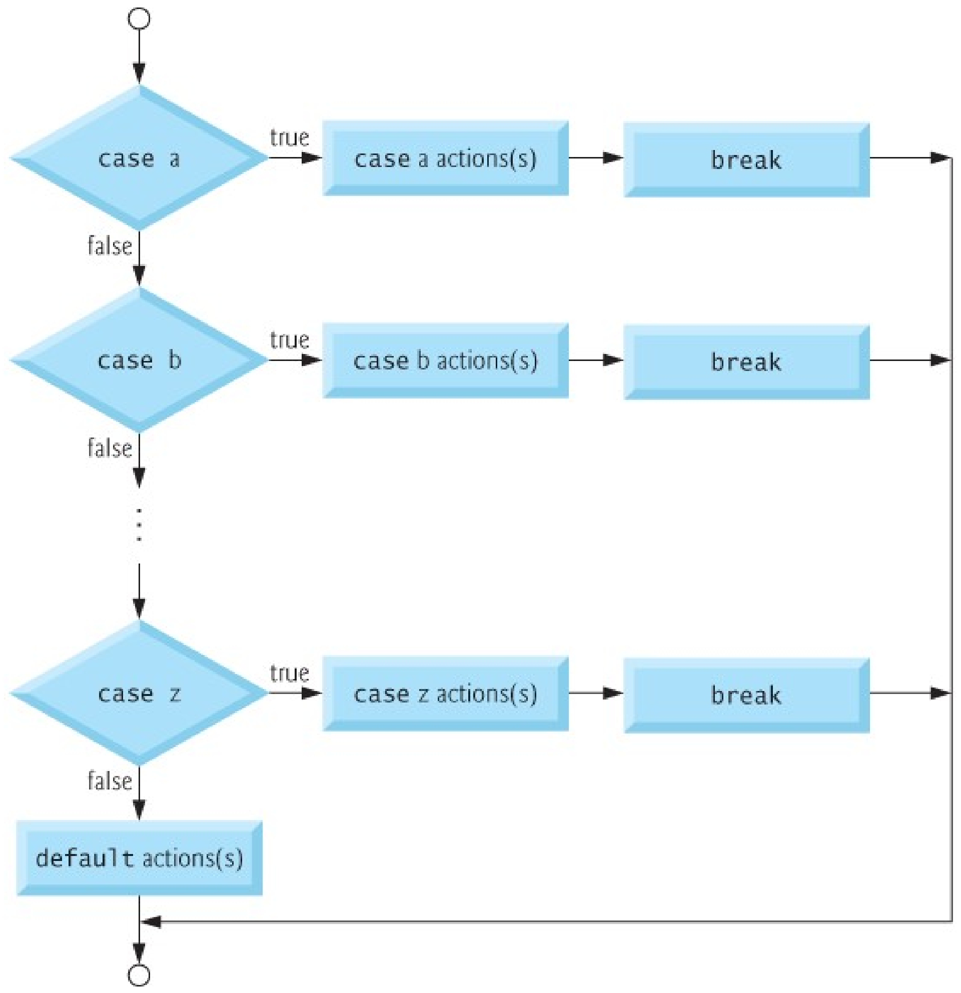
```
switch (…)  {

  case label:
          action(s) to take if controlling expression matches case;
          break;

  …

}
```

SWITCH STATEMENT FLOWCHART



NESTED IF VS. SWITCH STATEMENT

Nested if – more general

switch – can be easier to read

Case labels in switch cannot be doubles or strings, only integers or characters

SWITCH STATEMENT – EXAMPLE

Fig 4.7_simplified.c

Grade entered as a character

Grade is evaluated, and message printed

Line 11 has 2 parts:

- Read **char** from input using scanf function
- Store the char in **int** variable grade

Integral Data

Integral Types and Expressions

Remember, case labels can only contain **constant integral values**

Any combination of **character constants** and **integer constants** that evaluates to a **constant integer value**

So:

- A **single character in single quotes**, or
- An **integer**

Integral Data Types in C

Several ways to represent integers:

Data Type	Range of values
short int (or just **short**)	–32767 to +32767
long int (or just **long**)	–2,147,483,647 to +2,147,483,647
int	Between short and long; Usually the same as long
signed char	–127 to +127 Chars in computer's character data set
unsigned variations of all	0, and the positive end of the range

Storing a Single Byte

A single **byte** can be stored as a char or an integer

char version will store its character representation: ‘a’

int version will store its ASCII code representation: 97

Use format control string **%c** when reading chars using scanf

Confusing the Equality and Assignment Operators

Equality and Assignment Operators

Equality is == (think “equal to” or “equal two”)

Assignment is =

Using assignment in a condition is a logic error, not a compilation error

equalityOperatorError.c

After you write a program, search for “ = “ and verify that it’s used properly. This can help you prevent subtle bugs.

THE FFLUSH FUNCTION

FFLUSH() ISN'T RELATED TO SELECTION

But it's needed in the homework, so we're going to talk about it now.

When a number is read from stdin, we use the scanf function to interpret the bytes in the input stream as numbers

Example:

```
int x;              //declares the int variable x

scanf("%d", &x);    //reads bytes from stdin and
                    //stores them as an int in x
```

But Remember How scanf Works

It waits for the Enter Key to be pressed before processing stdin

The Enter Key is a character, not a number

It is not processed when a scanf reads a number.

That Enter Key character is still hanging around at the beginning of the input stream.

If the very next scanf in the program also tries to read the bytes as a number, it will ignore (and discard) that Enter Key and everything works fine.

But If the Next Scanf Tries to Read a Character…

If the very next scanf in the program tries to read the bytes as a character, the first character it encounters is the Enter Key.

fflush.c

To Solve This Problem – Use fflush After Every scanf

Format: **fflush(stdin);**

Note that there are 2 lower-case f's at the beginning.

This “flushes” the input stream of any lingering Enter Key characters.

This is required when reading a character after reading a number.

It’s not needed when only reading numbers.

You can add **fflush(stdin);** after each statement in your program that reads what the user types, just to be safe.