UNIVERSITY of LOUISVILLE
dare to be great

C Chapter 3: Conditions – Decision Making

CECS130
Introduction to Programming Languages
Dr. Roman V. Yampolskiy

Decision Statements (also known as Selection Statements)

- How to compare data values?

  Relational operators

- How to alter the sequence of program execution based on the result?

  if.. else statements

- How to deal with multiple choices?

  Switch statement

Motivational Example

- A professor wants to assign a letter grade based on percentage points.
- Percentages and the corresponding letter grades are shown in the table below.

Percent Range	>=90	>=80 < 90	>=70 <80	>=60 <70	<50
Letter Grade	A	B	C	D	F

Relational Operators

Operator	Meaning
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not Equal to

- **Relational expressions** evaluate to the integer values 1 (true) or 0 (false).
- We will use relational operators to form relational expression of conditions.

Three Logical Operators

- Exclamation mark **!**
  - NOT (negation)
  - unary
- Two ampersands **&&**
  - AND (conjunction)
  - binary
- Two vertical pipes **||**
  - OR (inclusive disjunction)
  - Binary

- These operators are used to combine more than one condition forming a complex condition.

Logical AND Operator

- Combines two relational expressions
- Combined expression true only if BOTH expressions are true

true	&&	false	false
false	&&	true	false
false	&&	false	false
true	&&	true	true

Example Use of &&

```
if ( age < 1  &&  gender == 'm')
{
    printf ("Infant boy\n") ;
}
```

Logical OR Operator

- Combines two relational expressions
- Combined expression false only if BOTH expressions are false

true	||	false	true
false	||	true	true
false	||	false	false
true	||	true	true

Example Use of ||

```
if (grade == 'D'  ||  grade == 'F')
{
        printf ("See you next semester!\n") ;
}
```

Logical NOT Operator

- Reverses result of relational expression
- Example: ! (x == y)

x	y
7	7

Evaluate relational expression, does 7 equal 7?

! (true) so negates to false

Example Use of !

```
if ( ! (x == 2) )   /* same as  (x != 2)  */
{
    printf("x is not equal to 2.\n") ;
}
```

Recall: Precedence table

	Description	Represented By
1	Parenthesis	() []
1	Structure Access	. ->
2	Unary	! ~ ++ -- - * &
3	Mutiply, Divide, Modulus	* / %
4	Add, Subtract	+ -
5	Shift Right, Left	>> <<
6	Greater, Less Than, etc	> < =
7	Equal, Not Equal	== !=
8	Bitwise AND	&
9	Bitwise Exclusive OR	^
10	Bitwise OR	|
11	Logical AND	&&
12	Logical OR	||
13	Conditional Expression	?:
14	Assignment	= += -= etc
15	Comma	,

Practice with Relational Expressions

int a = 1, b = 2, c = 3 ;

Expression	Value
a < c	True
b <= c	True
c <= a	False
a > b	False
b >= c	False

More Practice:

Assume: a = 4, b = -2, and c = 0

x = (a > b || b > c && a == b)

x = ((a > b) || (b > c) && (a == b))

x = ((4 > -2) || (-2 > 0) && (4 == -2))

x = (TRUE || (FALSE && FALSE))

x = (TRUE || FALSE)

x = (TRUE)

x = 1

Arithmetic Expressions: True or False

- **Arithmetic expressions** evaluate to numeric values.
- An arithmetic expression that has a value of zero is false.
- An arithmetic expression that has a value other than zero is true.

if Statement

- An if statement allows a program to choose whether or not to execute a following statement.
- Syntax: (structure/format)

if (condition)

statement;

- Semantics: (meaning)

Condition is a Boolean expression

Something that evaluates to True or False.

If condition is true then execute the statement.

The **if** statement: syntax

```
if (expression)
        statement;        //single statement executed
                          //if expression is true

 if (expression)
 {                        //statements inside {} are
                          //executed if expression is true
        statement1;
        statement2;
        ...
        statement n;
 }
```

Examples

```
if ( age >= 18 )
    printf(“Vote!\n”);
```

```
if ( value == 0 )
{
    printf (“The value you entered was zero.\n”) ;
    printf (“Please try again.\n”) ;
}
```

if/else Statement

- An if-else statement allows a program to do one thing if a condition is true and a different thing if the condition is false.
- Syntax:

```
if ( condition )
        statement1
else
        statement2
```

- Statements to be executed for if and else can be a single statement or multiple statements enclosed in { }.

Example

```
if ( age >= 18 )
{
    printf(“Vote!\n”) ;
}
else
{
    printf(“Maybe next year!\n”) ;
}
```

Another Example

```
if ( value == 0 )
{
    printf (“The value you entered was zero.\n”) ;
    printf(“Please try again.\n”) ;
}
else
{
    printf (“Value = %d.\n”, value) ;
}
```

Nesting of if-else Statements

```
if ( condition1 )
{
    statement(s)
}
else if ( condition2 )
{
   statement(s)
}
     . . .                  /* more else clauses may be here */
else
{
   statement(s)      /* the default case */
}
```

Nested if-else Example

```
if ( value == 0 )
{
    printf (“The value you entered was zero.\n”) ;
}
else if ( value < 0 )
{
   printf (“%d is negative.\n”, value) ;
}
else
{
   printf (“%d is positive.\n”, value) ;
}
```

Good Programming Practice

- Always place braces around the body of an if statement.
- Advantages:
  - Easier to read
  - Will not forget to add the braces if you go back and add a second statement to the body
  - Less likely to make a semantic error
- Indent the body of the if statement 3 to 4 spaces -- be consistent!

The ***switch*** Statement

- Another way to evaluate expressions
- Frequently used to create a menu
- Reduces the need for multiple if/else
- Makes for cleaner, easier to debug code

The **switch** statement: Syntax

```
 switch (expression)
{
      case constant:
                  statement(s);
                  break;
      case constant:
                  statement(s);
                  break;
       /* default is optional*/
      default:
                  statement(s);
}
```

The **switch** statement

- *Expression* must be of type integer or character
- The keyword **case** must be followed by a *constant*
- **break** statement is required unless you want all subsequent statements to be executed.

Practice!

Convert these nested **if/else** statements to a **switch** statement:

```
if (rank==1 || rank==2)
   printf("Lower division \n“);
else
{
      if (rank==3 || rank==4)
         printf("Upper division \n“);
      else
      {    if (rank==5)
                  printf("Graduate student \n“);
                  else
                              printf("Invalid rank \n“);
      }
}
```

Practice Solution!

```
switch(rank)

{
   case 1: case 2:
     printf("Lower division \n“);
     break;
   case 3: case 4:
     printf("Upper division \n“);
     break;
   case 5:
     printf("Graduate student \n“);
     break;
   default:
     printf("Invalid rank \n“);
}//end switch
```

The Conditional Operator

- Shorthand version of if-then statement
- The ?: operator is used as follows:

expression1?expression2: expression3

where if expression1 is true, then the result is expression2, otherwise it is expression3;

a>b?1:3.5

if a>b then 1, else 3.5

= versus ==

```
int a = 2 ;
if ( a = 1 ) {  /* semantic (logic) error! */
    printf (“a is one\n”) ;
}
else if ( a == 2 ) {
    printf (“a is two\n”) ;
} else {
    printf (“a is %d\n”, a) ;
}
```

Output is:

a is one

= versus ==

- The statement if (a = 1) is syntactically correct, so no error message will be produced.
  - (Some compilers will produce a warning.)
  - However, a semantic (logic) error will occur.
- An assignment expression has a value (the value being assigned).
- In this case the value being assigned is 1, which is true.
- If the value being assigned was 0, then the expression would evaluate to 0, which is false.
- This is a VERY common error. So, if your if-else structure always executes the same, look for this typographical error.

Input Validation: Upper/Lower Case?

- Make sure your program is user friendly
- If user is asked to choose A or B and he selects b your program should gracefully handle it
- Instead of: **if (response == ‘A’) {…}**
- Do: **if ((response == ‘A’) || (response == ‘a’)) {…}**

Input Validation: Range of Values

```
int response = 0;

printf("Enter a number between 1 and 10");
scanf("%d, &response);

if (response <1 || response >10) {
    printf("\nNumber not in range.\n");
} else
printf("\nThank you.\n");
```

Input Validation: isdigit() function

- Sometimes necessary to check if input is a number
- C provides function isdigit(x) if you #include <ctype.h>
  - x is input from the user
- Returns true if x is a number, false otherwise

```
printf("\nPlease enter a number between 1 and 9\n");
scanf("%c", &cResponse);
if isdigit(cResponse)
    printf("\nThank you.\n");
else printf("\n You didn't enter a digit…\n");
```

Random Numbers

- Make your program produce different output each run
- Great for games, simulations, experiments, encryption, gambling software
- Computer software generates pseudo random numbers
- C provides an easy to use built-in function: rand()

Random Numbers

- To generate random numbers between 0 and x -1
  - iRandomNumber = rand() % x
- To generate random numbers between 1 and x
  - iRandomNumber = (rand() % x) + 1
- For Example: random numbers from 1 to 10
  - iRandomNumber = (rand() %10) + 1
- To generate a different sequences of random numbers every time your program runs
  - Call srand() function

Random Numbers

- A call of the srand() function takes the following format:
  - **srand(seed);**
  - Seed is a whole number used to generate a random sequence of numbers
  - Current time (in seconds) is a good candidate to be a seed
    - Obtained via function time()
    - Requires you to #include <time.h> library
  - **srand(time(NULL));**

Chapter Example

```
#include <stdio.h>
#include <time.h>
main() {
	int iRandomNum = 0;
	srand(time(NULL));		//seed the random number generator with time
	iRandomNum = (rand() % 4) + 1; //generate a random number between 1 and 4 inclusive

	// display a message corresponding to the random choice
	switch (iRandomNum) {
		case 1:	printf("\nYou will meet a new friend today.\n");  break;
		case 2:	printf("\nYou will enjoy a long and happy life.\n"); break;
		case 3:	printf("\nOpportunity knocks softly. Can you hear it?\n"); break;
		case 4:	printf("\nYou'll be financially rewarded for your good deeds.\n"); break;
	} //end switch

	printf("\nLucky lotto numbers: "); // print 6 random lucky numbers between 1 and 49 inclusive
	printf("%d ", (rand() % 49) + 1);
	printf("%d ", (rand() % 49) + 1);
	printf("%d ", (rand() % 49) + 1);
	printf("%d ", (rand() % 49) + 1);
	printf("%d ", (rand() % 49) + 1);
	printf("%d\n", (rand() % 49) + 1);
}
```

Summary

- In many applications, choices are to be made depending on some conditions related to the problem.
- Selection or decision structures are used to model such situations.
- C/C++ supports the implementation of “selection” through the “if”/”else” and “switch” statements.
- Alternatively random decision can be made.

The End!