Lecture-2
Conditional Statements

Dr. Hacer Yalım Keleş

Ref: Programming in ANSI C, Kumar

CONDITIONAL EXPRESSION OPERATOR

The *conditional expression operator,* unlike all other operators in C that are either unary or binary, is a ternary operator and takes three arguments. It has the following operator formation:

expression-1 ? *expression-2* : *expression-3*

where the question mark ? and the colon : are the two symbols that denote this operator. A conditional expression is evaluated by first evaluating *expression-1*. If the resultant value is nonzero *(true),* then *expression-2* is evaluated and the value of *expression-2* becomes the result of the conditional expression. Otherwise, *expression-3* is evaluated and its value becomes the result.

```
larger = x > y ? x : y;
```

assigns to larger the value of x if x is greater than y; otherwise, it assigns the value of y to larger.

For example, the conditional expression

```
c?x = a:x = b
```

would be interpreted as

```
(c ? x = a : x) = b
```

precedence of conditional expr. is lower than all other operators except the assignment and coma operators!

due to the lower precedence of the assignment operator. This assignment is illegal, since the conditional expression produces an rvalue. A correct form of this conditional expression is

```
c ? x = a : (x = b)
```

Note that it is not necessary to put parentheses around the second operand x = a. Of course, a better way to write this expression is

```
x = c ? a : b
```

The conditional expression operator is right-associative with respect to its first and third operands, so that

a ? b : c ? d : e

is interpreted as

a ? b : (c ? d : e)

The value of this expression would be b if a is nonzero *(true)*; otherwise, the value of the expression would be d if c is nonzero *(true)* and e if c is zero *(false)*.

CONDITIONAL STATEMENTS

A conditional statement allows selective processing of a statement or a group of statements. There are two forms of conditional statements: the `i f` statement and the `if-else` statement.

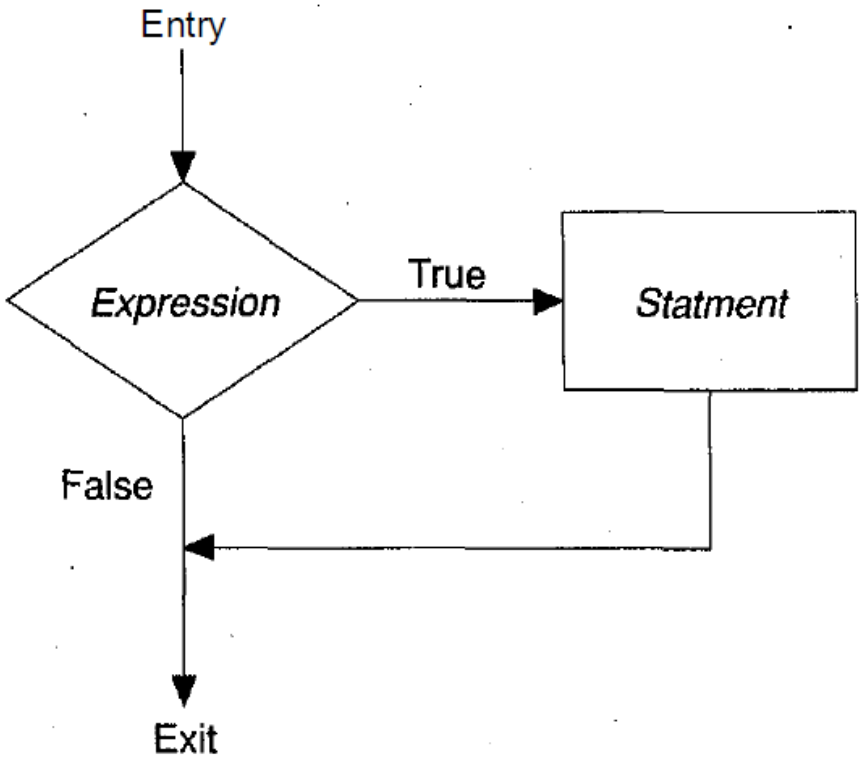
if Statement

The `if` statement is used to specify conditional execution of a program statement, or a group of statements enclosed in braces. The general format of this statement is

`if (` *expression* `)`

statement

When an `if` statement is encountered, *expression* is evaluated and if its value is nonzero *(true)*, then *statement* is executed. After the execution of *statement*, the statement following the `if` statement is executed next. If the value of *expression* is zero *(false)*, *statement* is not executed and the execution continues from the statement immediately after the if statement.



`if` statement

Here are some examples:

1.
```
if (number < 0)
    number = -number;
printf("%d\n", number);
```

2.
```
if (age >18 && salary < 250)
  {
    unemployed++;
    total_age += age;
    total_salary += salary;
  }
```

```c
#include <stdio.h>

int main(void)
  {
    int v1, v2, larger;

    scanf("%d %d", &v1, &v2);

    larger = v1;
    if (v2 > v1) larger = v2;

    printf("%d\n", larger);

    return 0;
  }
```

if-else Statement

```
if ( expression )
        statement-1
else
        statement-2
```

When an `if-else` statement is encountered, the value of *expression* is evaluated and if its value is nonzero *(true)*, *statement-1* is executed. After the execution of *statement-1*, the program execution continues from the statement immediately after *statement-2*. If the value of *expression* is zero *(false)*, *statement-2* is executed. After the execution of *statement-2*, the program execution continues from the statement following *statement-2*. In either case, one of *statement-1* or *statement-2* is executed, but not both.

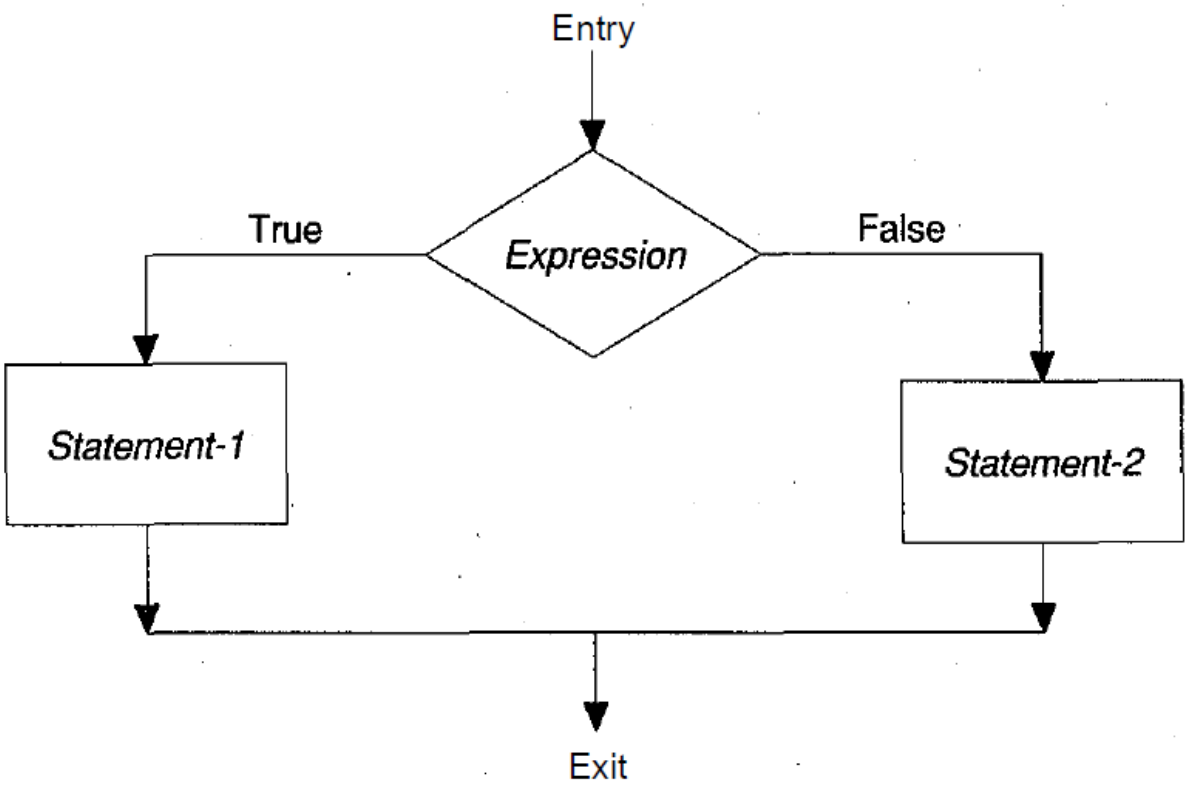
Here are some examples:

```
1. if (n % 2 == 0)
       n = even;
   else
       n = odd;
```

can equivalently be written using the conditional expression operator as

```
n = (n % 2 == 0) ? even : odd;
```

```
2. if (classification == star_hacker)
     {
       regular_pay = 2 * regular_rate * regular_hrs;
       overtime_pay = 5 * regular_rate * overtime_hrs;
     }
   else
     {
       regular_pay = regular_rate * regular_hrs;
       overtime_pay = 2 * regular_rate * overtime_hrs;
     }
```

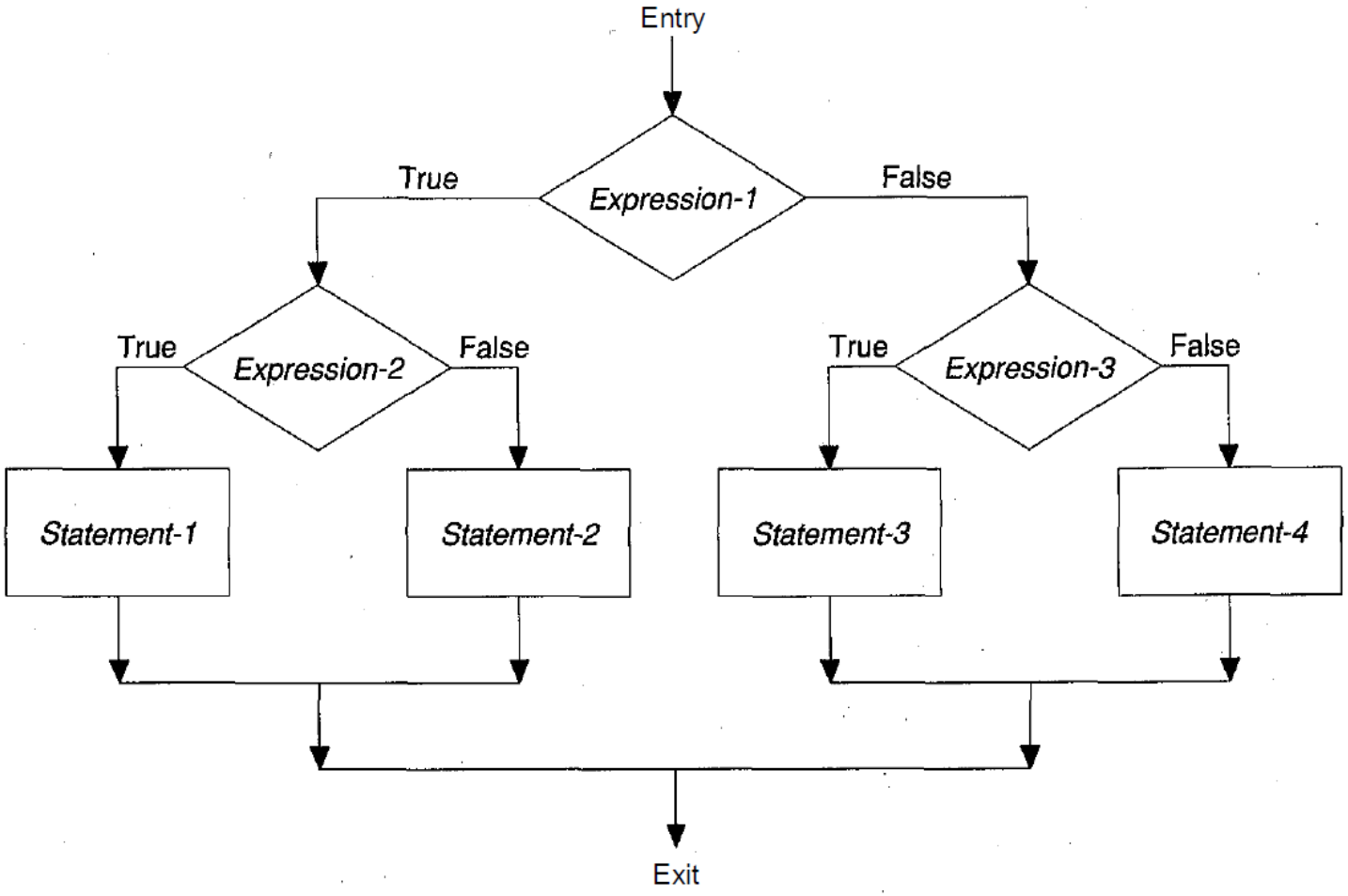


`if-else` statement

NESTED CONDITIONAL STATEMENT

In our discussion of the if-else statement, we identified a set of statements as the if-block and another set as the else-block. No restrictions were placed on the kinds of statements that could be included within these blocks. One could, therefore, include another conditional statement in either the i f-block or the else-block or both. If this were to happen, the resulting statement is called a *nested conditional* statement.

```
if ( expression-1 )
        if ( expression-2 )
            statement-1
      else
            statement-2
else
        if ( expression-3 )
            statement-3
      else
            statement-4
```



Nested conditional statement

```
if (x1 > x2)
    if (x1 > x3)
        largest = x1;
    else
        largest = x3;
else
    if (x2 > x3)
        largest = x2;
    else
        largest = x3;
```

This statement assigns the largest of `x1`, `x2`, and `x3` to `largest`.

Sequence of Nested ifs

Consider a sequence of nested `if` s as in

```
if ( expression-1 )
        if ( expression-2 )
                if ( expression-3 )
                                .
                                .
                                .
                        if ( expression-n )
                                statement
```

meaning thereby that *expression-2* should be evaluated only if *expression-1* is *true, expression-3* should be evaluated only if both *expression-1* and *expression-2* are *true,* and so on, and *statement* should only be executed if all the expressions are *true.* This nested-conditional statement can equivalently be written as

```
if ( expression-1 && expression-2 && ... && expression-n )
                statement
```

Dangling else Problem

```
if ( expression-1 )
        if ( expression-2 )
            statement-1
    /* no else-block fortheinner if-else statement */
else
    statement-3
```

is not interpreted as intended by the indentation format. Instead, `else` is associated with the closest previous else-less `if`, and the statement is interpreted as

```
if ( expression-1 )
        if ( expression-2 )
            statement-1
    else
            statement-3
```

The difficulty in such cases in which an innermost if does not contain an `else`, but an outer `if` does, can be alleviated by the use of braces for proper association. The braces have the effect of closing the `if` statement. Thus, we can write

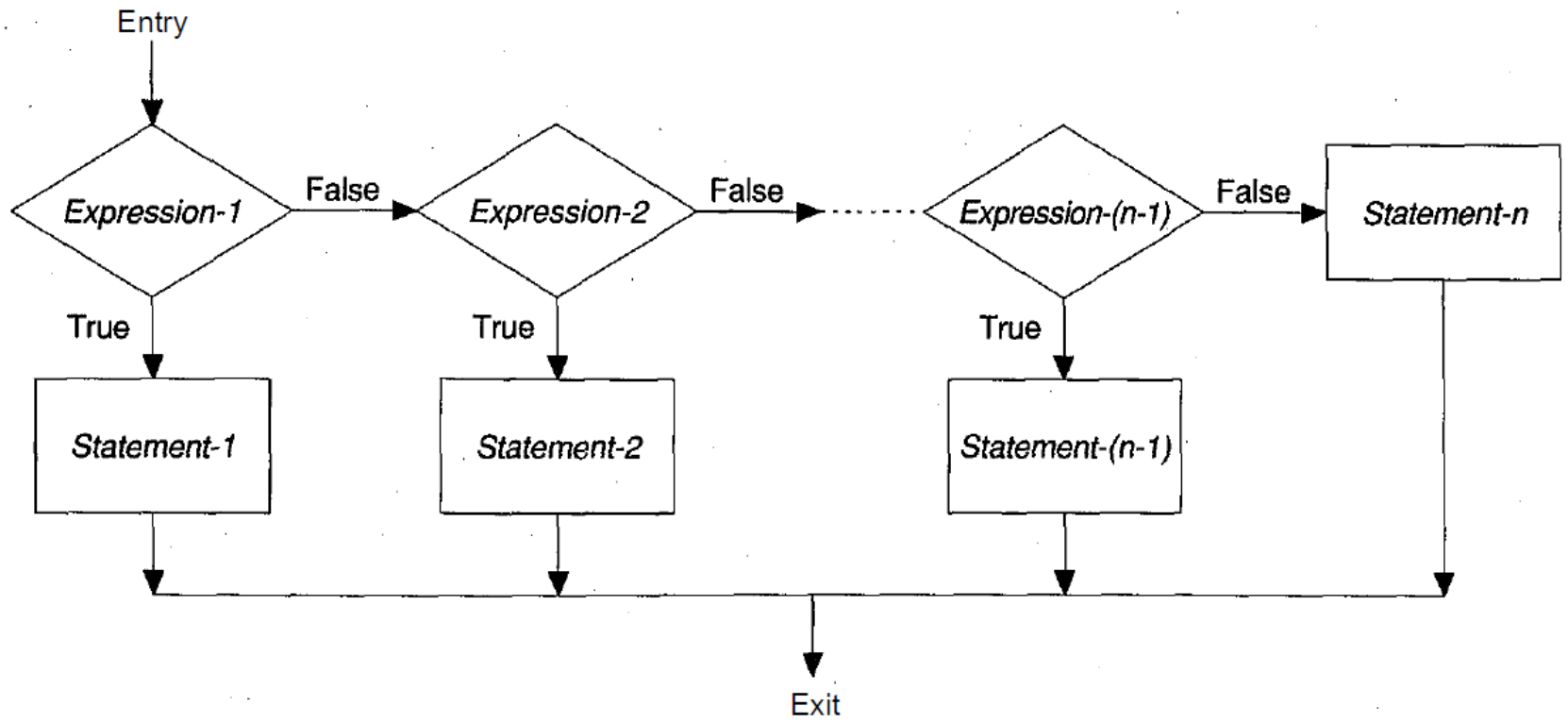
```
if ( expression-1 )
    {
        if ( expression-2 )
            statement-1

    }
else
    statement-3
```

for the desired effect.

MULTIWAY CONDITIONAL STATEMENT

```
if ( expression-1 )
      statement-1
else if ( expression-2 )
      statement-2
else if ( expression-3 )
      statement-3
        .
        .
        .
else if ( expression-(n-1) )
      statement-(n-1)
else
      statement-n
```

In a multiway conditional statement, conditional expressions are evaluated in order. If any of these expressions is found to be *true,* the statement associated with it is executed, and this terminates the whole chain. If none of the expressions is *true* the statement associated with the final `else` is executed.



Multiway conditional statement

CONSTANT MULTIWAY CONDITIONAL STATEMENT

When each of the tests in a multiway if statement checks for a different value of the same expression, we have a *constant multiway decision,* which is coded using the `switch` statement. The general format of the `switch` statement is as follows:

```
switch ( expression )
{
     case value-1 :
          statement-1
          break;
     case value-2 :
          statement-2
          break;
               .
               .
               .
     case value-n :
          statement-n
          break;
     default :
          statement-x
          break;
}
```

Expression must be an integral expression and the `case` values must be constant integral expressions.

The preceding `switch` statement is equivalent to the following `if-else` statement:

```
if ( expression == value-1 )
        statement-1
else if ( expression == value-2 )
        statement-2
                .
                .
                .
else if ( expression == value-n )
        statement-n
else
        statement-x
```

```c
♦include <stdio.h>

int main(void)
  {
    char operator;
    float operandl, operand2;

    scanf("%f %c %f", &operandl, &operator, &operand2);

    switch (operator)
      {
        case '+':
            printf("%f\n", operandl + operand2);
            break;
        case '-!
            printf("%f\n", operandl - operand2);
            break;
        case '*' :
            printf("%f\n", operandl * operand2);
            break;
        case '/' :
            printf("%f\n", operandl / operand2);
            break;
        default:
            printf("Invalid Operator\n");
            break;
      }

    return 0;
  }
```

Following is the program for the same problem using a multiway `if-else` statement, instead of a `switch` statement:

```
♦include <stdio.h>

int main(void)
  {
    char operator;
    float operandl, operand2;

    scanf("%f %c %f", Soperandl, Soperator, &operand2);

    if (operator == '+')
        printf("%f\n", operandl + operand2);
    else if (operator == '-')
        printf ("%f\n", operandl - operand2);
    else if (operator =='*')
        printf("%f\n", operandl * operand2);
    else if (operator == '/')
        printf("%f\n", operandl / operand2);
    else
        printf("Invalid Operator\n");

    return 0;
  }
```