Rajarata University of Sri Lanka

COM 1407
Computer Programming

LECTURE 05 – **FLOW CONTROL STRUCTURES – DECISION MAKING**

BY: PIYUMI HERATH

Objectives

- At the end of this lesson students should be able to :
- Understand that all coding languages use common concepts like conditionals.
- Understand that conditionals are statements that are carried out when certain criteria are met.
- Evaluate a conditional statement and predict the outcome, given an input.
- Write conditional statements, defining criteria for when a program should take certain actions.

Decision making constructs

- C languages is equipped with specific statements that allow us to check a condition and execute certain parts of code depending on whether the condition is true or false. Such statements are called conditional, and are a form of composite statement.
- In C there are three forms of conditional statements
  - The if statement
  - The conditional operator
  - The switch statement

1.if statements

The if statement

- The C programming language provides a general decision-making capability in the form of a language construct known as the if statement.
- The general format of this statement is as follows:

  if (*expression*)
  program statement;

  if (this condition is true)
  execute this statement ;

- Imagine that you could translate a statement such as "If it is not raining, then I will go swimming" into the C language.

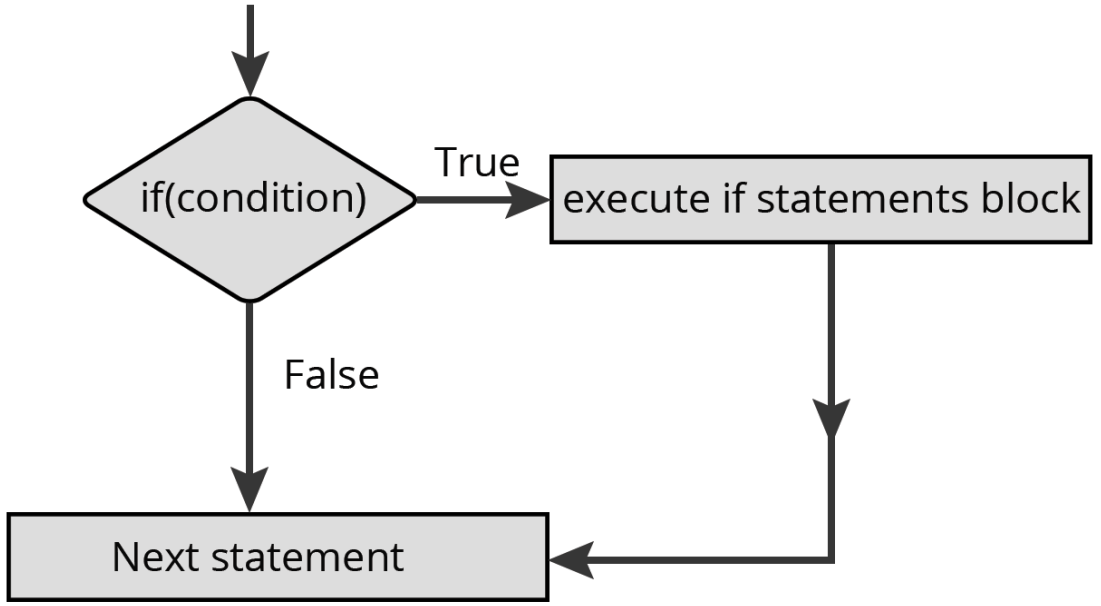
  Using the preceding format for the if statement, this might be "written" in C as follows:

  if (it is not raining)
  I will go swimming

- The if statement is used to stipulate execution of a program statement (or statements if enclosed in braces) based upon specified conditions

Semantics:

- The keyword **if** tells the compiler that what follows is a decision control instruction.
- The condition following the keyword **if** is always enclosed within a pair of parentheses. If the condition, whatever it is, is true, then the statement is executed.
- If the condition is not true then the statement is not executed; instead the program skips past it.



Example

```
include <stdio.h>

int main( )
{
int num ;
printf ( "Enter a number less than 10 " ) ;
scanf ( "%d", &num ) ;
if ( num <= 10 )
        printf ( "What an obedient servant you are !" ) ;
}
```

The printf statement is executed *only* if the value of num is less than or equal to the value of 10; otherwise, it is ignored.

Expressing conditions

The condition in an if statement can be an arbitrary expression of type Boolean.

- 'Relational' operators. The relational operators should be familiar to you except for the equality operator **==** and the inequality operator **!=**. Note that **=** is used for assignment, whereas, **==** is used for comparison of two quantities.
- A variable of type `boolean`;

  Example:

```
boolean finished;
...
if (finished)...
```

- A complex boolean expression, obtained by applying the boolean operators `!`, `&&`, and `||` to simple expressions

1.2 The if-else Construct

- The if-else is actually just an extension of the general format of the if statement.
- The if-else statement allows us to select between two alternatives
- Syntax:

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

Semantics

- If the result of the evaluation of expression is TRUE, program statement 1, which immediately follows, is executed;
- otherwise, program statement 2is executed.
- In either case, either program statement 1or program statement 2is executed, but not both

We can use multiple if statements to check more than one conditions.

```
#include <stdio.h>
int main()
{    int x, y;
    printf("enter the value of x:");
    scanf("%d", &x);
    printf("enter the value of y:");
    scanf("%d", &y);
    if (x>y)
       {      printf("x is greater than y\n");    }
    if (x<y)
       {      printf("x is less than y\n");     }
    if (x==y)
       {      printf("x is equal to y\n");    }
    printf("End of Program");
    return 0;
}
```

- The `else` part of an `if-else` statement is optional.
- If it is missing, we have an `if` statement, which allows us to execute a certain part of code if a condition is satisfied (and do nothing otherwise).

Activity

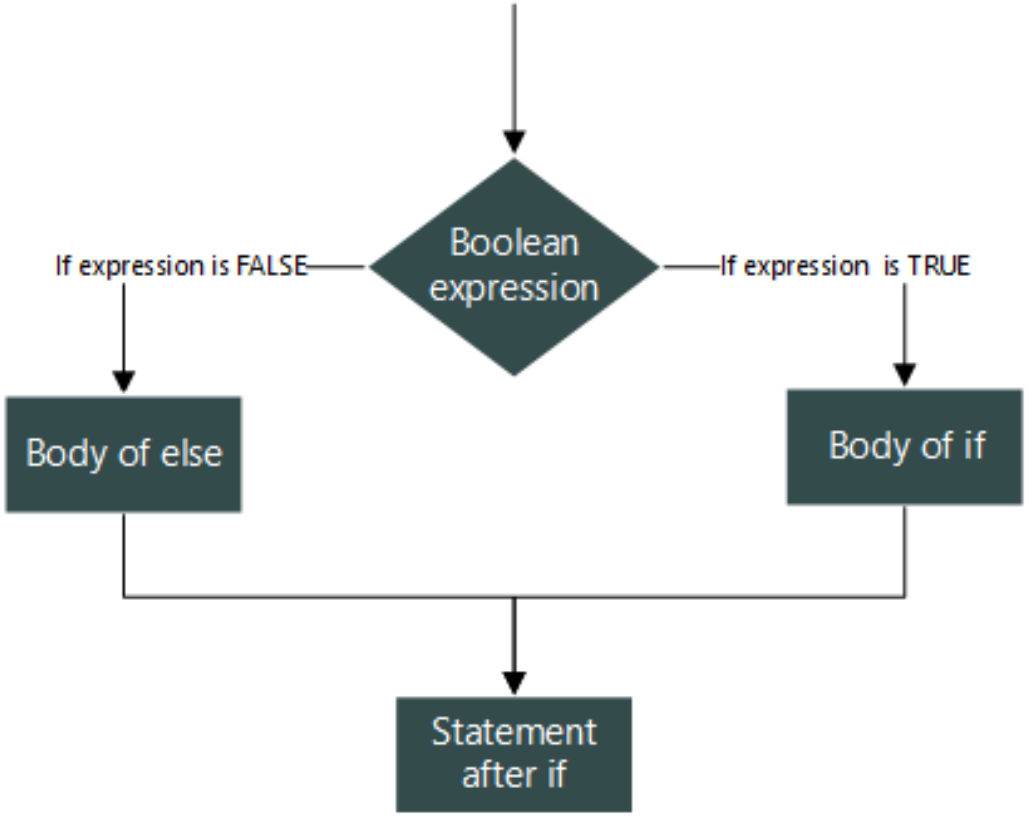
- **1.** Write a C program to accept two integers and check whether they are equal or not.
- Test Data : 15 15
  Expected Output :
  Number1 and Number2 are equal

Answer :

```
#include <stdio.h>
void main()
{
    int int1, int2;
    printf("Input the values for Number1 and Number2 : ");
    scanf("%d %d", &int1, &int2);

    if (int1 == int2)
        printf("Number1 and Number2 are equal\n");
    else
        printf("Number1 and Number2 are not equal\n");
}
```

Flow chart


Boolean expression
If expression is FALSE
If expression is TRUE
Body of else
Body of if
Statement after if

1.3 Nested if-else

- We can write an entire **if-else** construct within either the body of the **if** statement or the body of an **else** statement. This is called 'nesting'of **if**s.

```
int main( )
{
        int i ;
        printf ( "Enter either 1 or 2 " ) ;
        scanf ( "%d", &i ) ;

        if ( i == 1 )
                printf ( "You are in line 1!" ) ;
        else
        {
                if ( i == 2 )
                        printf ( "You are in line 2" ) ;
                else
                        printf ( "Incorrect input" ) ;
        }
}
```

- Note that the second **if-else** construct is nested in the first **else** statement. If the condition in the first **if** statement is false, then the condition in the second **if** statement is checked. If it is false as well, then the final **else** statement is executed.
- In the above program an **if-else** occurs within the **else** block of the first **if** statement. Similarly, in some other program an **if-else** may occur in the **if** block as well. There is no limit on how deeply the **if**s and the **else**s can be nested

1.4 If- else-if construct

- Programming decisions that you have to make are not always so black-and-white (two - fold)
- This method of formatting improves the readability

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

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

Use of indentation

- If the body of an IF statement is just a single statement, then there isn't a need for a set of braces surrounding it.
- By placing the (indented) body on the next line, you make the whole statement (and also its purpose) much easier to identify:

```
if (count < 0)
        count = 0;
```

- **IF with a Compound Body :** Indent the statements of the body uniformly and stick with your adopted brace-placement style.

```
if ( (!victor(human)) && (!victor(computer)) ) {
        number_of_draws++;
        printf("We are well-matched.\n");
}
```

IF-ELSE with Simple Bodies

- The same rules apply here as they apply to the ordinary IF. Braces are not necessary.

```
if (temperature < 55)
        printf("It could be warmer...\n");
else
        printf("It could be colder...\n");
```

- Here's the same example with that brace alignment:

```
if (temperature < 55) {
        printf("It could be warmer...\n");
} else {
        printf("It could be colder...\n");
}
```

IF-ELSE with Compound Bodies

Of course, in this case the braces are required. Just be consistent with your style.

```
if (victor(human))
{
        human_wins++;
        printf("Human wins.\n");
}
else{
        computer_wins++;
        printf("Computer wins !\n");
}
```

2. Conditional Operator

Conditional Operator

- The conditional operators **?** and **:** are sometimes called ternary operators since they take three arguments. In fact, they form a kind of foreshortened if-then-else. Their general form is,

  expression 1 ? expression 2 : expression 3

- "if expression 1 is true (that is, if its value is non-zero), then the value returned will be expression 2, otherwise the value returned will be expression 3".
- The ternary operator provides a shorthand way of writing the *if...else* statements. The ternary operator is represented by the question mark (?) symbol and it takes three operands**: a condition to check, a result for true, and a result for false.**

Cont.

```
int x, y ;
scanf ( "%d", &x ) ;
y = ( x > 5 ? 3 : 4 ) ;
```

This statement will store 3 in y if x is greater than 5, otherwise it will store 4 in y.
The equivalent if statement will be,

```
if ( x > 5 )
        y = 3 ;
else
        y = 4 ;
```

Cont.

- It's not necessary that the conditional operators should be used only in arithmetic statements. This is illustrated in the following examples:
- Ex.:

```
int i ;
scanf ( "%d", &i ) ;
( i == 1 ? printf ( "Amit" ) : printf ( "All and sundry" ) ) ;
```

- Ex.:

```
char a = 'z' ;
printf ( "%c" , ( a >= 'a' ? a : '!' ) ) ;
```

- The conditional operators can be nested as shown below.

```
int big, a, b, c ;
big = ( a > b ? ( a > c ? 3: 4 ) : ( b > c ? 6: 8 ) ) ;
```

Cont.

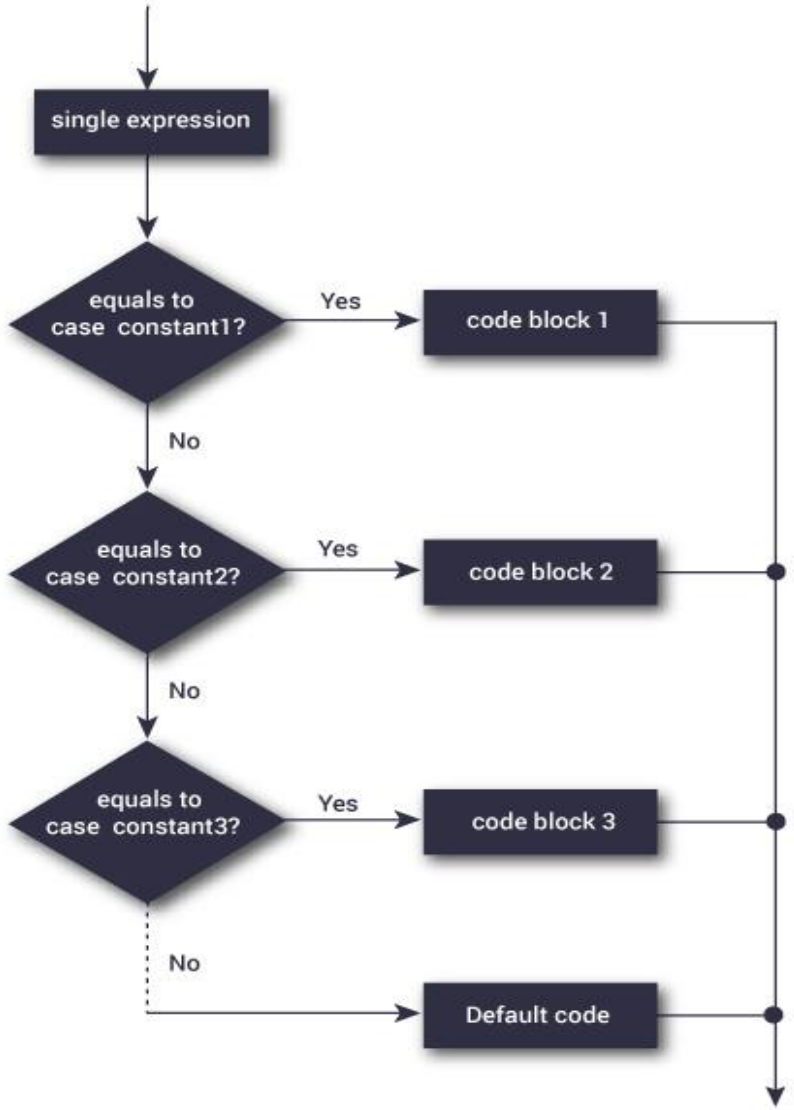
Check out the following conditional expression:

- a > b ? g = a : g = b ; This will give you an error 'Lvalue Required'. The error can be overcome by enclosing the statement in the : part within a pair of parenthesis.
- This is shown below: a > b ? g = a : (g = b) ;
- In absence of parentheses the compiler believes that b is being assigned to the result of the expression to the left of second =. Hence it reports an error.
- The limitation of the conditional operators is that after the ? or after the : only one C statement can occur. In practice rarely is this the requirement. Therefore, in serious C programming conditional operators aren't as frequently used as the if-else.

3. Switch Statements

Switch statements

- Switch statements are a more efficient way to code when testing multiple conditions.
- In switch statements value of a variable is successively compared against different values.



Syntax

```
switch (<Expression>)
{
case <value 1>:
        <statement 1.1>;
        <statement 1.2>;
        break;
case <value 2>:
        <statement 2.1>;
        <statement 2.2>;
        break;
…
case <value n>:
        <statement n.1>;
        <statement n.2>;
        break;
default:
        <statement D.1>;
        <statement D.2>;
}
```

<Expression> is compared with each of the case values **<value 1>, <value 2>, <value 3>** etc.

- If a match is found, the statements that correspond to the first match, and all the other case statements further down are executed

– For example if **<Expression> is equal to <value 2>, <statement2.1>, <statement2.2>** etc. are executed; then, **<statement3.1>, <statement3.2>** etc. are executed, and so on.

- If a match is not found, default statements **<statementD.1>, <statementD.2>** etc. are executed.
- The default statements are optional. If it is not there, case statement completes without doing anything.

- **<Expression>** is compared with each of the case values **<value 1>, <value 2>, <value 3>** etc.
  If a match is found, the statements that correspond to the first match, and all the other case statements further down are executed
  - For example if **<Expression> is equal to <value 2>, <statement2.1>,<statement2.2>** etc. are executed; then, **<statement3.1>, <statement3.2>** etc. are executed, and so on.
- If a match is not found, default statements **<statementD.1>, <statementD.2>** etc. are executed.
- The default statements are optional. If it is not there, case statement completes without doing anything.

- At times we may want to execute a common set of statements for multiple **case**s
-

```
switch ( ch )
{
case 'a' :
case 'A' :
  printf ( "a as in apple" ) ;
  break ;
case 'b' :
case 'B' :
  printf ( "b as in brain" ) ;
  break ;
default :
  printf ( "wish you knew what are alphabets" ) ;
}
```

- If we have no **default** case, then the program simply falls through the entire **switch** and continues with the next instruction (if any,) that follows the closing brace of **switch**.
- The disadvantage of **switch** is that one cannot have a case in a **switch** which looks like:
  case i <= 20 :
  All that we can have after the case is an **int** constant or a **char** constant or an expression that evaluates to one of these constants. Even a **float** is not allowed.
- The advantage of **switch** over **if** is that it leads to a more structured program and the level of indentation is manageable, more so if there are multiple statements within each **case** of a **switch**.
- We can check the value of any expression in a **switch**. Thus the following **switch** statements are legal.
  switch (i + j * k)
  switch (a < 4 && b > 7)
  Expressions can also be used in cases provided they are constant expressions. Thus **case 3 + 7** is correct, however, **case a + b** is incorrect.

Thank you.

Next Lesson :

Flow Control Structures – Loop Control