

Fundamentals of Programming

Lecture 5

Chamila Karunatilake

Department of Information and Communication Technology

Faculty of Technology

University of Sri Jayewardenepura

chamilakarunatilake@sjp.ac.lk

Lecture Outline

- C Expressions , Statements and Blocks
- Introduction to C control flow structures
- C Branching
 - If condition statements (if, else, else if)
 - Nested if statements
- Switch Statements
- Conditional Operator

C Expressions

- In C Language, an expression is any legal combination of **one or more explicit values, constants, variables, operators, and functions** that **represents a value**.
- The representing value (resulting value) is usually one of various primitive types or derived data types.
- A value or a variable all by itself is also considered an expression.

Examples:

- `x+5`
- `42`
- `a`
- `a = 12`

“An assignment expression has the value of the left operand after the assignment”

C Statements

- An expression such as `x = 0` or `i++` or `printf ("Hello World!\n")` **becomes a statement** when it is followed by a **semicolon**.

```
x = 0;  
i++;  
printf ("Hello World!\n") ;
```

- Semicolon itself is considered as a valid statement which does nothing.

```
;
```

C Blocks

- A pair of braces { }, are used to group statements together into a compound statement, or block.

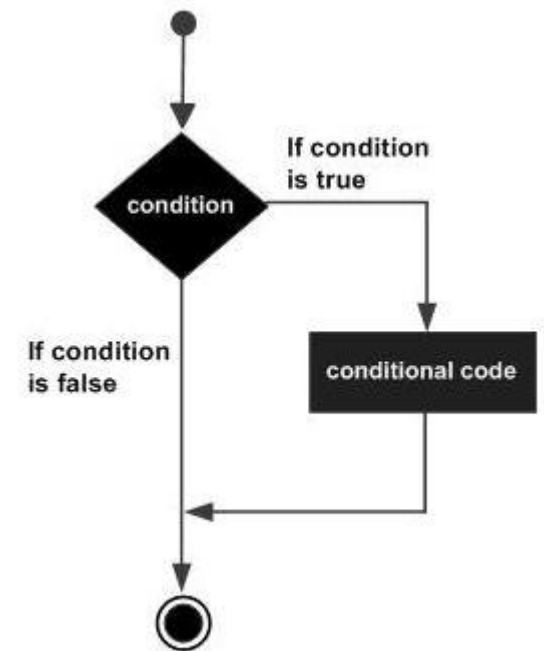
```
{  
    int x=19;  
    int y = 10;  
    printf ("%d %d\n", x++, --y) ;  
}
```

C Control Structures

- C programs which were discussed up to now have a regular flow, from first line to last line of the program.
- C language(and many other languages) provide mechanisms to handle alternative, non-linear flows of the program.
- These flow controlling mechanisms are called **flow control structures** and they specify the order in which computations are performed.
- C provides two styles of flow control:
 - Branching
 - Looping
- **Branching** specifies what actions to be taken and **looping** states how many times a certain action to be repeated.

Branching Control Structures

- Using branching statements, the flow of the program is divided into two or more branches and depending on a condition specified at the branching, only one of those paths are followed.
- There are three main categories of branching data structures
 - If condition statements(including if, else, else if)
 - switch statements
 - Conditional operator(?:)



if Condition Statements

- Basic if statement has **if** keyword followed by a Boolean condition statement which is placed between a pair of opening and closing parentheses.

`if(condition)`

- That condition statement is usually a Boolean expression, that means the result value is true or false.

`if(x>10)`

`if(x==0)`

`if(x<=y)`

`if(x != 0)`

- However, any expression could be used since there is a Boolean value representation for every expression. If the value is something like 0, 0.0 or null, the Boolean value is false and for any other nonzero value the Boolean value is true.

`if(0)`

`if(1)`

`if('C')`

`if('\0')`

- if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

if Condition Statements


- If statement is followed by a single line statement or a block of statements. The block is indicated by a pair of braces.

```
if (x<0)
x++;
```


```
if (x<0)
{
    printf ("%d" ,x) ;
    x++;
}
```

- Only If the condition is true, the following statement or the block of statements are executed.
- In a case of the condition is false, the following statement or the block of statements are skipped and jumped into the subsequent statements.

```
if (x<0)
x++;
```



```
if (x<0)
{
    printf ("%d" ,x) ;
    x++;
}
```



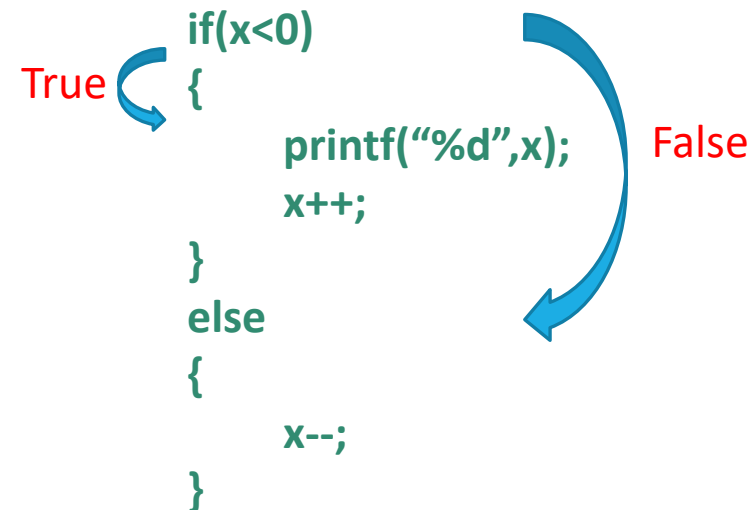
if Condition Statements : Example

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int value;
    printf("Enter a negative or positive Integer : ");
    scanf("%d",&value);
    if(value < 0)
    {
        value*=-1;
    }
    printf("The Absolute value is : %d\n",value);
    return 0;
}
```

if else Condition Statements

- In basic if statement, when condition is not satisfied (false), the execution is jumped back into the regular flow skipping the if statement or block.
- There are situations that two branches are necessary. Execution follows one branch or the other, depending on the condition is true or false.
- The only difference from regular “if statement” is that immediately after if block’s closing brace(or statement’s semicolon in a case of a single statement), “**else**” key word and a block of statements would be placed.

```
if(condition)
{
    .....
}
else
{
    .....
}
```



if else Condition Statements

```
#include <stdio.h>

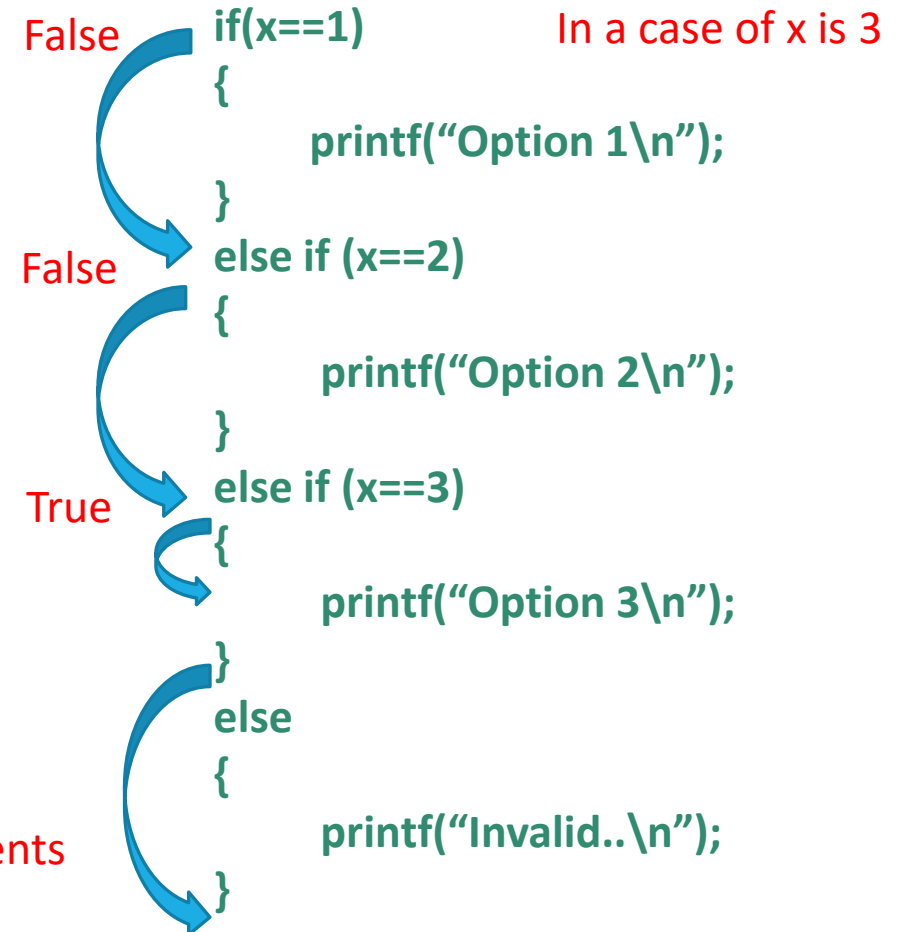
int main()
{
    int value;
    printf("Enter a Positive Integer : ");
    scanf("%d",&value);
    if(value < 0)
    {
        printf("Invalid input...!");
    }
    else
    {
        printf("The value you have entered : %d\n",value);
    }
    return 0;
}
```

If, else if, else Condition Statements

- In case of many parallel branches are needed in program execution, C provides **else if** statements.
- Following an if block, instead of else block, one or more **else if** blocks could be used to specify several branches of execution flows.
- Optional **else** block would follow **else if** blocks and program only executes the **else if** block that satisfies the condition.
- If no **else if** block is satisfied the condition, it will check for an **else** block at the end of the **else if** blocks.
- If **else** is found, **else** block is executed. If there is no **else** block, then execution moved back to the normal flow and the rest of the program is executed.

If, else if, else Condition Statements

```
if(condition 1)
{
    .....
}
else if (condition 2)
{
    .....
}
else if (condition 3)
{
    .....
}
else
{
    .....
}
```



If, else if, else Condition Statements : Example

```
#include <stdio.h>
#define GUESS_NUM 27
int main()
{
    int number;
    printf("Guess a number between 0 and 100...\n");
    printf("Enter your guess : ");
    scanf("%d", &number);
    if(number == GUESS_NUM)
    {
        printf("Congratulations!\n");
        printf("You have correctly guessed the answer...! \n");
    }
    else if(number < GUESS_NUM)
    {
        printf("The Answer is greater than your guess...\n");
    }
    else if(number > GUESS_NUM)
    {
        printf("The Answer is less than your guess...\n");
    }
    else
    {
        printf("Invalid input");
    }
    return 0;
}
```

Nested *if* statements

- Inside if, else or else if block, another if, if-else or if-elseif-else blocks can be included.
- These types of conditional statements are called nested if statements. The internal if statements are independent from conditions of the outer if statements.

```
if(condition 1)
{
    if(condition 2)
    {
        .....
    }
}
else
{
    .....
}
```

```
if(a==100)
{
    if(b==100)
    {
        printf("Both are 100!" );
    }
}
else
{
    printf("A is not 100!" );
}
```


Nested *if* statements: Example

```
#include <stdio.h>

int main () {

    int a = 100;
    int b = 200;
    if( a == 100 ) {
        if( b == 200 ) {
            printf("Value of a is 100 and b is 200\n" );
        }
    }
    printf("Exact value of a is : %d\n", a );
    printf("Exact value of b is : %d\n", b );
    return 0;
}
```

switch Statements

- The switch statement is a multi-way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly.
- Switch statement is very similar to else if statement.
- It can be used in special situations instead of **else if** where an **expression** is checked whether it is **matched** one of the several **constant** values.

Example:

```
if(x==1)
    printf("One\n");
else if (x==2)
    printf("Two\n");
else if (x==3)
    printf("One\n");
```



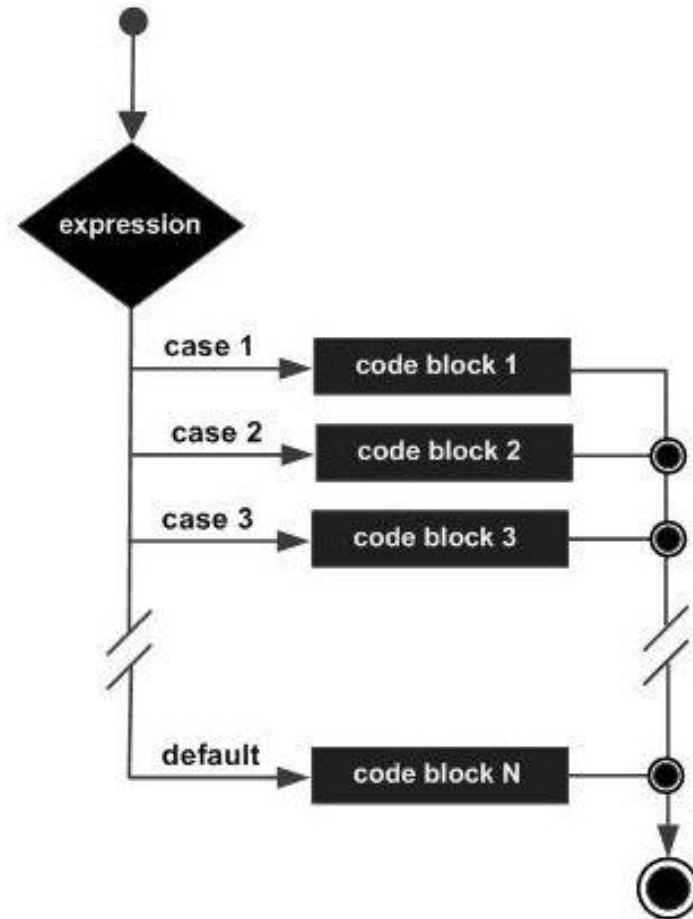
```
switch(x)
{
    case 1:
        printf("One\n");
        break;
    case 2:
        printf("One\n");
        break;
    case 3:
        printf("One\n");
        break;
}
```

switch Statements

- In a switch statement, there are several parts, **switch**, **case(s)**, statements, **break**, and **default**-case.
- switch part states the expression to be compared with several cases and each case is labeled by one or more **integer-valued constants** or **constant expressions**.
- If a case matches the expression value, execution starts at that case.
- All case expressions must be different.
- The case **default** is executed if none of the other cases are satisfied, and it is optional
- Inside case, syntactically **optional break** can be placed. The break statement causes an immediate exit from the switch otherwise the execution goes through to the next case block.

```
switch(expression) {  
    case constant-expression :  
        statement(s) ;  
        break;  
    case constant-expression :  
        statement(s) ;  
        break;  
    default :  
        statement(s) ;  
}
```

switch Statements



switch Statements: Example

```
#include <stdio.h>

int main () {
    int option;
    printf("Enter option 1,2,or 3 to select your language: " );
    scanf("%d",&option);
    switch(option) {
        case 1 :
            printf("You have selected Sinhala language\n" );
            break;
        case 2 :
            printf("You have selected Tamil language\n" );
            break;
        case 3 :
            printf("You have selected English language\n" );
            break;
        default :
            printf("Invalid Option...\n" );
    }
    return 0;
}
```

switch Statements: Example

```
#include <stdio.h>

int main () {
    char grade;
    printf("Enter your grade : " );
    scanf("%c",&grade);
    switch(grade) {
        case 'A' :
            printf("Excellent!\n" );
            break;
        case 'B' :
        case 'C' :
            printf("Well done!\n" );
            break;
        case 'D' :
            printf("You passed!\n" );
            break;
        case 'F' :
            printf("Better try again\n" );
            break;
        default :
            printf("Invalid grade...\n" );
    }
    printf("Your grade is  %c\n", grade );
    return 0;
}
```

Conditional Operator

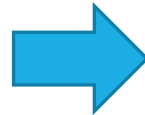
- The conditional operator can be used as a **shorthand** for some if-else statements.

- The general syntax of the conditional operator is:

expression1 ? expression2 : expression3

- This is an expression, **not a statement**, so it represents a **value**.
- The operator works by evaluating expression1. If it is true (non-zero), it evaluates and returns expression2 . Otherwise, it evaluates and returns expression3.

```
if (x < y) {  
    min = x;  
}  
else {  
    min = y;  
}
```



```
min = (x < y) ? x : y;
```

Conditional Operator

```
#include <stdio.h>

int main () {
    int val1, val2, min;
    printf("Enter an integer : " );
    scanf("%d", &val1);
    printf("Enter another integer : " );
    scanf("%d", &val2);
    min = (val1 < val2) ? val1 : val2;
    printf("Minimum value is : %d\n", min );
    return 0;
}
```

Questions?