UNIT 4. CONTROL FLOW

Programming
Year 2017-2018
Grade in Industrial Technology Engineering

Paula de Toledo.
David Griol

Universidad
Carlos III de Madrid
www.uc3m.es

Contents

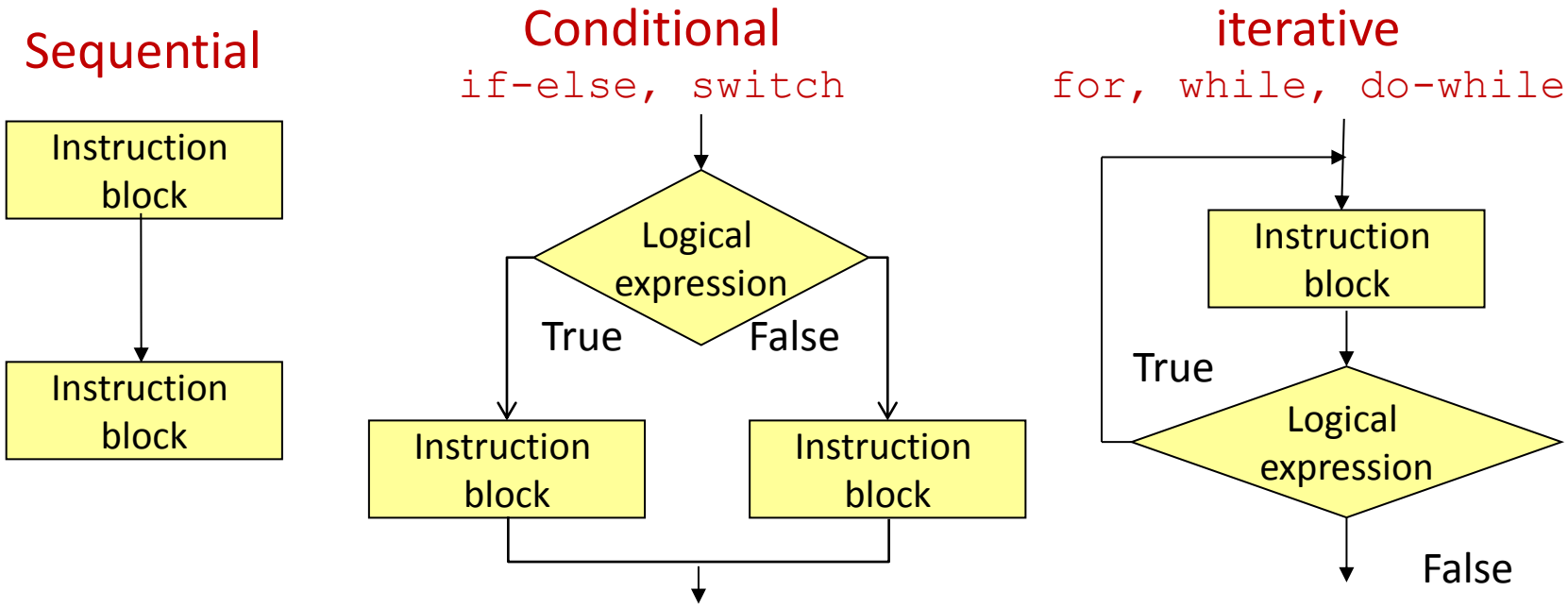
Control flow structures

- Alter the standard flow of program execution
  - Standard = Starting from the first instruction of the main method, sequential order

- Control flow instructions break up this sequence
  - **Conditional control flow structures**
    - Decision-making instructions
    - Blocks of instructions are executed depending on the result of a boolean expression (the condition)
  - **Repetitive (Iterative) control flow structures (AKA Loops)**
    - Blocks of instructions are repeated while a condition holds

4

Every possible algorithm can be implemented using only these three control flow structures



Structured programming

- Programing paradigm
  - Best practice for developing good programs
    - Good = easy to develop and to maintain (correct, upgrade)

- Basic principles of structured programming
  - Single entry and exit end point (start /end)
  - Only secuential, conditional and interative control flow structures allowed

- Never use “go to” instructions!

6

2. CONDITIONAL INSTRUCTIONS
2.1 IF-ELSE

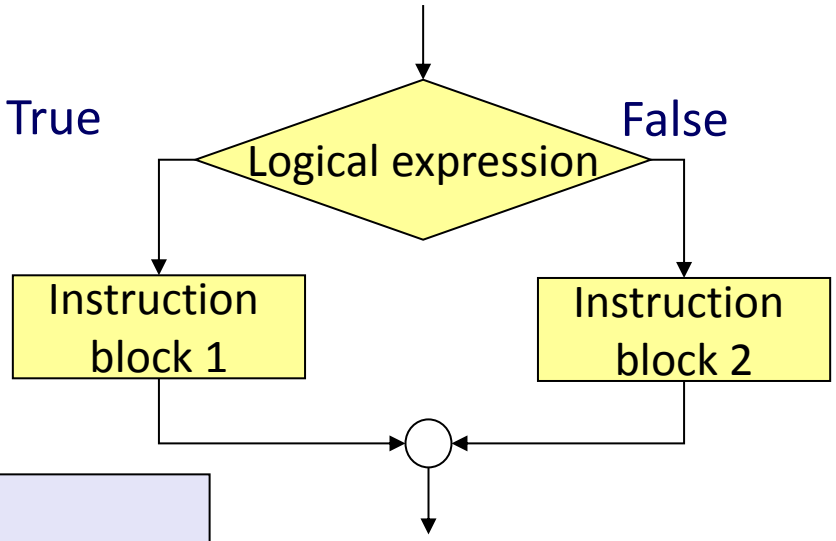
Conditional structure `if - else`

- Logic expression is evaluated
  - If the expression is true block of code 1 is run
  - If the expression is false block of code 2 is run
  - After either branch has been executed, control returns to the point after the if

Syntax

```
if (Logical expression){
        instruction_block_1;
}
else {
        instruction_block_2;
}
```

Flow diagram



Example

```
if (a > b){
   printf("A greater than B");
} else {
   printf("A smaller than or equal to B");
}
```

Conditional structure `if`

- Simplest version
- If the expression is true the block is run
- If it is not, nothing happens

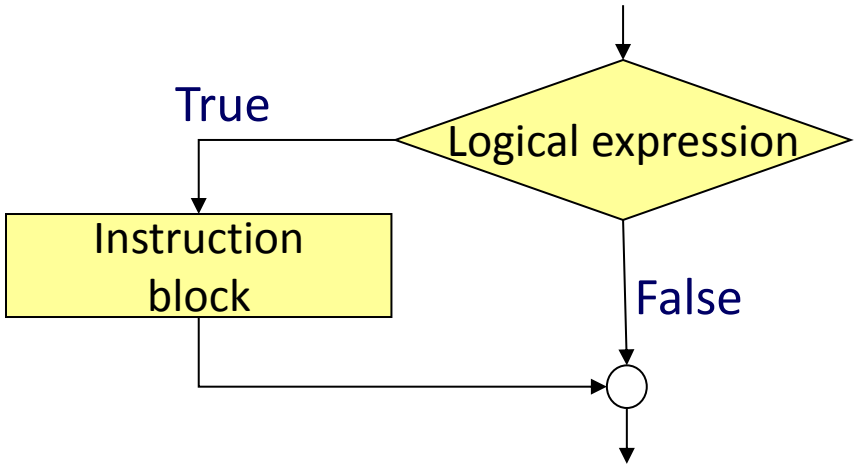
Syntax

```
if (Logical expression) {
    instruction_block;
}
```

Example

```
if (age < 18){
   printf("KID");
   ticket_discount = 20;
}
```

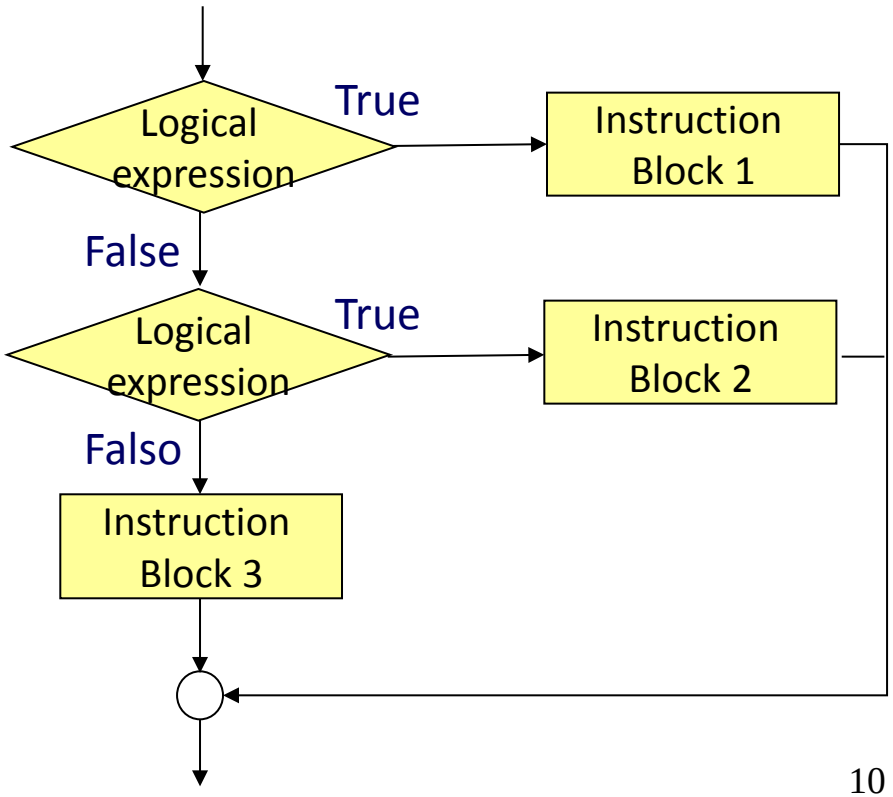
Flow diagram



Nested if structures

- To define diferent alternative and mutually exclusive paths
- If all logical expressions are false the last block is run

```
if (logical_expression_1){
        instruction_block_1;
}
else{
    if(logical_expression_2){
       instruction_block_2;
    }
    else {
       instruction_block_3;
    }
}
```



Exercise

- Develop a program that queries the user for the mark in one exam and displays the corresponding grade
  - Sobresaliente: 9 to 10
  - Notable: 7 to 9
  - Bien: 5 to 7
  - Insuficiente: less than 5
  - Error –the mark is not between 0 and 10

Example

```c
#include <stdio.h>
int main(void)
{
  int mark; //student's mark - numeric value

  printf("Insert mark: (0-10) \n");
  scanf("%i", &mark);

  if ( (mark >= 0) && (mark < 5) ){
    printf("Failed\n");
  }
  else{
      if ( (mark >= 5) && ( mark <= 10 )){
        printf("Passed \n");
      }
      else{
        printf("mark value not valid");
        printf("valid range is 0-10\n");
      }
  }
  system("PAUSE");
  return 0;
}
```

Option 1

Option 2

Default option

2. CONDITIONAL INSTRUCTIONS
2.2. SWITCH

switch

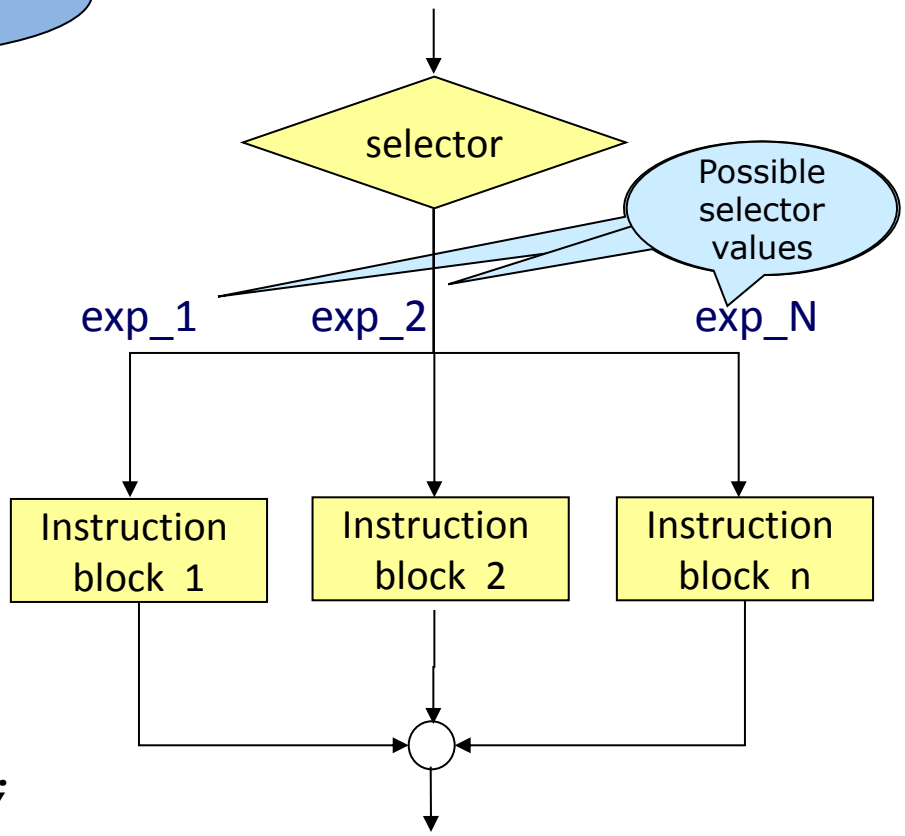
- Simplifies multiple selection structures based on a selector variable

Syntax

int o char variable

```
switch(selector){
   case constant_expression_1:
      instruction_block_1;
      break;
   case constant_expression_2:
      instruction_block_2;
      break;
   case constant_expression_N:
      instruction_block_N;
      break;
   ……
   default:
      default_instruction_block;
}
```

Flow diagram



Example: Tell the polygon name according to the number of sides

```c
#include <stdio.h>

int main(void)
{
  int numSides;
  printf ("Insert number of sides: ");
  scanf("%i", & numSides);
  switch (numSides){ //numSides is the selector for the switch
    case 0: case 1: case 2:
        printf("not a polygon \n");
        break;
    case 3:
        printf("triangle\n");
        break;
    case 4:
        printf("rectangle\n");
        break;
    case 5:
        printf("pentagon\n");
  }
  system("PAUSE");
  return 0;
}
```

switch

- selector:
  - Must be a variable or expression of any of the following datatypes: integer, logical, char
    - Can't be a real number (float, double)
    - Avoid using a logical expression, in that case "if" is a better option
- case
  - Each "case" clause is followed by a constant expression of the same datatype as the selector
    - Examples 12, MAXIMUM, MAXIMUM+5
  - Value ranges cant be set, a new case clause needed for each value
    - Example case 1: case 2: case 3: case 4
    - Restriction in C, this is possible in most languages
- default:
  - If the selector value doesn't match the value of any of the case clauses, the default block will be run (if existing)

Example: Tell if the letter entered is a vowel

```
int main(void){
  char c;
  printf ("Enter a letter :");
  scanf("%c", &c);
  switch (c){
     case 'A': case 'a':
        printf ("vowel A\n");
        break;
     case 'E': case 'e':
        printf (" vowel E\n");
        break;
     case 'I': case 'i':
        printf (" vowel I\n");
        break;
     case 'O': case 'o':
        printf (" vowel O\n");
        break;
     case 'U': case 'u':
        printf (" vowel U\n");
        break;
     default: //Default block - run if no previous match
        printf ("consonant\n");
    }
  system("PAUSE");
  return 0;
}
```

Switch: break instruction

- Usually we want all the case blocks in a switch to be mutually exclusive
  - To get this behaviour we end each case block with a break instruction
  - If the break instruction is missign, all the following case blocks from that point on will be run (fall through behaviour) switch a until a break instruction, is found
- When a break instruction is found, the switch is terminated

Switch with no break, example

- Occasionally we may want to write some code that falls through the different options of the switch
- Example: Madrid anti-pollution protocol: three different scenarios (1-2-3) according to severity, increasing traffic restrictions

```
int scenario;
printf("Enter the scenario ");
scanf("%i",&scenario);

switch(scenario){
    case 3:
       printf("Half of cars banned, according to plate number");
    case 2:
       printf("On-street parking banned for cars with no resident permit");
    case 1:
       printf("Maximum speed in M30 70Kmh");
       break;
    case default:
       printf("No traffic restriction ");
}
```

3. ITERATIVE STRUCTURES - LOOPS

Iterative control flow structures

- Other names: repetitive structures, loops
- Three options in C
  - `for`
    - The instruction block is repeated a given number of times, that is known beforehand (when the loop starts)
      - Example :
        - Display all numbers from 1 to 100.
  - `while` and `do-while`
    - The instruction block is repeated while a given condition holds
    - Used when the number of iterations is not known a priori (for example depends on the user inputs)
    - Example :
      - Read pin number until pin is correct

For control flow structure

Syntax

```
for (initialization; control_expression; update) {
        instruction_block;
}
```

Initialization

An initial value is assigned to the de control variable

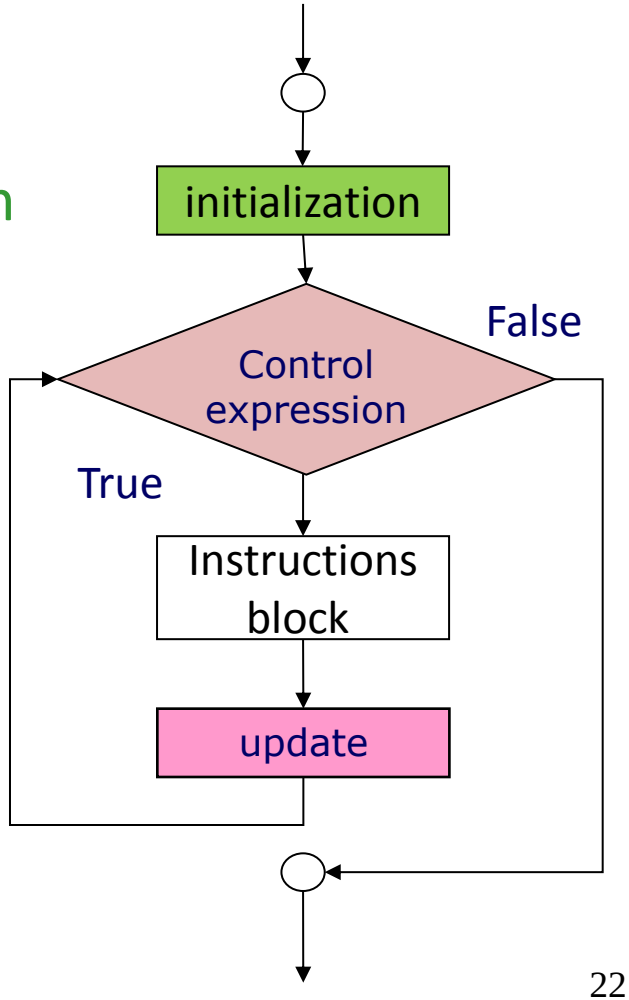
Control expression:

Boolean (logical) expression that is checked before each loop iteration and determines if the block is run once more or not

Update

Update of the control variable performed after each loop iteration

Flow diagram



`For` control flow structure

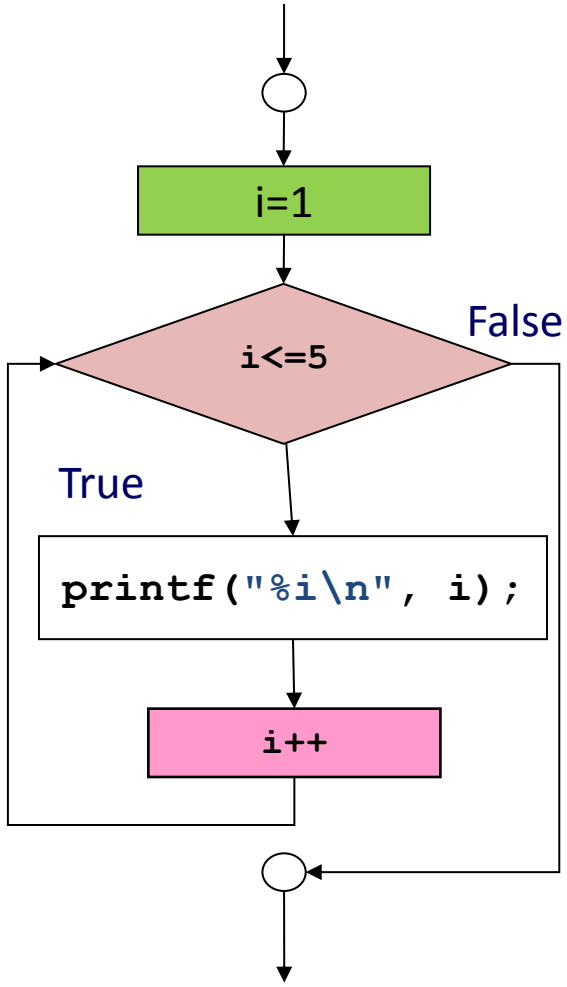
- Repeats an instruction block a given number of times

- For loops comprise
  - An initialization instruction
    - Executed before the first iteración (only)
  - An update instruction
    - Updates the value of the control variable, executed after every iteration
  - A control expression, that is evaluated after the update
  - If the control expresion is true the block of instructions is executed again

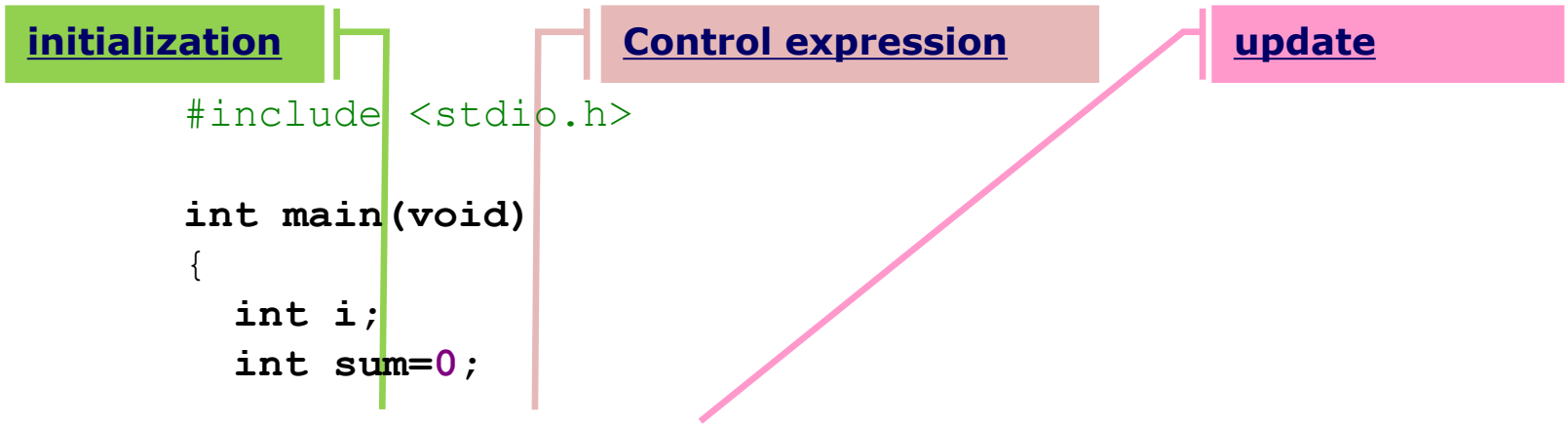
For Example 1

- Program displaying all integer numbers from 1 to 5

```
#include <stdio.h>
int main(void) {
    int i;
    for (i=1; i<=5; i++){
        printf("%i\n", i);
    }
    return 0;
}
```



For: Program to add all integers from 1 to 10




```
#include <stdio.h>

int main(void)
{
  int i;
  int sum=0;

  for (i=1; i<=10; i++){
     sum=sum+i;
     printf ("i variable's value now is %i \n", i);
     printf ("sum value is now %i \n", sum);
  }
  printf ("The final value of i is %i \n", i);
  printf (" The final value of sum is %i\n ", sum);

  system("PAUSE");
  return 0;
}
```

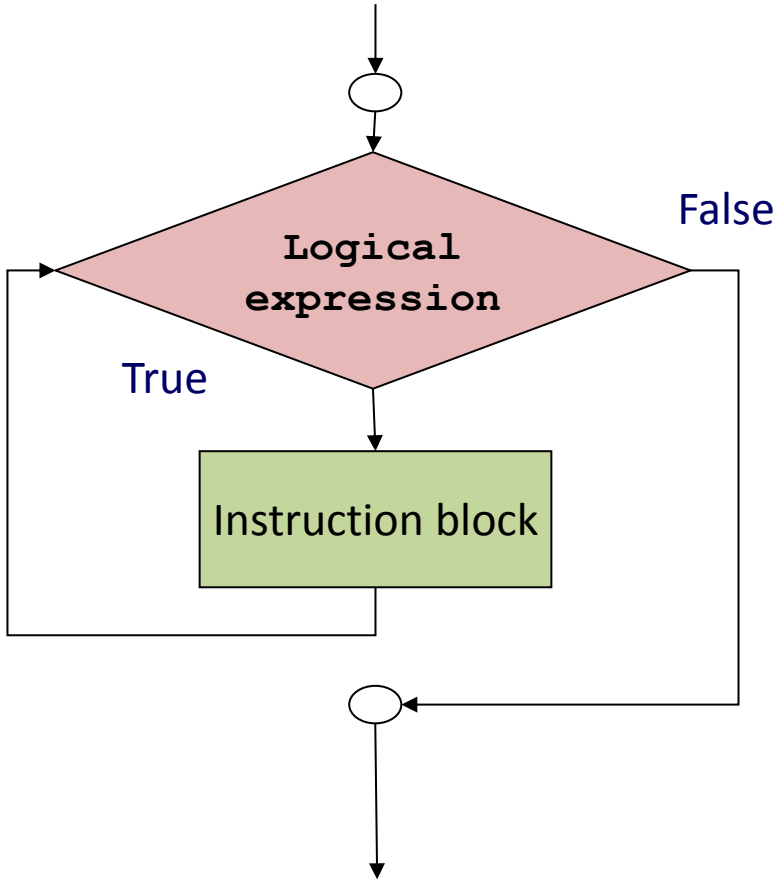
WHILE

`while` control structure

Syntax

```
while (logical_expresion) {
    instruction_block;
}
```

Flow diagram



`while` control structure

- Repeats a block of instruction **while** the logical expression (condition) is true
- The logical expression is evaluated **before** starting to run the instruction block
  - Therefore the **number of repetitions can be 0**.
  - After each execution of the instruction block the condition is re-evaluated.
  - If the condition is still true, the instruction block is repeated.
  - If the condition is now false, the loop terminates.
- We should check that the condition will be false under some situations
  - ... otherwise the loop will run forever (infinite loop)

Program to add all integers from 1 to 10 Is while the best option?

Program to add all numbers entered by the user until user enters a 0

Example: random number generator

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int num;
  int answer;

  printf ("Do you want to generate random numbers (1->YES 0->NO)");
  scanf ("%d", &answer);

  //The loop will only run if answer is 1
 while (answer==1){
     num=rand()%1024;
     printf ("%d \n", num);
     printf ("Do you want to generate more random numbers (1->YES 0->NO)? ");
     scanf ("%d", & answer);
  }

  printf ("You entered %d. No more random numbers will be generated\n", answer);
  system("PAUSE");
  return 0;
}
```

Exercises: Guess secret number

- For Beginners
  - Secret number is set in the code as a constant
  - Hints: "bigger than", "smaller than"
- Advanced user
  - Limited number of attempts
- Expert user
  - A second number is a "bomb" – if you hit the bomb game is over

3.3. DO-WHILE

Do – while control structure

- The instruction block is executed at least once

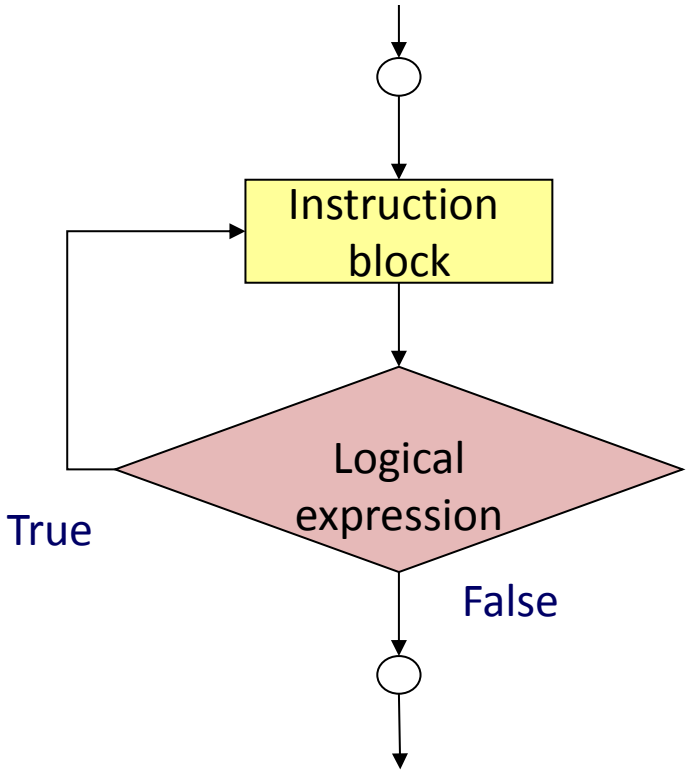
Sintaxis

```
do {

        instruction_block;

} while (logical_expression);
```

Flow diagram



Do – while control structure

- As in while, the instruction block is executed **while** a condition or logical expression is true
- Only difference: the logical expression is evaluated **after** the instruction block is executed.
  - Mínimum number of repetitions is one.
- After executing the instruction block the expression is evaluated again.
  - If the condition is still true, the instruction block is repeated.
  - If the condition is now false, the loop terminates

- Similarly to while, an infinite loop can be generated

do-while example

- Display menu, read option, until exit (0).

```
int main(void)
{
  int option; // no need to initialize option here

  do{
    printf ("Select one option\n");
    printf ("1: Add numbers\n");
    printf ("2: Subtract numbers\n");
    printf ("3: Multiply numbers\n");
    printf ("0: Exit\n");

      // --- code for operations here ----

    scanf ("%d", &option);
  } while (option!=0);  // the loop will stop only when option is 0

  printf ("You selected exit\n");
  system("PAUSE");
  return 0;
}
```

Program that prompts for a password until correct

```c
#include <stdio.h>
#define CORRECT_PASSWORD 1234

int main(void)
{
  int password;

  do{
    printf ("Enter your password: ");
    scanf ("%i", &password);

  }while (password!= CORRECT_PASSWORD);

  printf ("Welcome!\n");
  system("PAUSE");
  return 0;
}
```

Password + Limited number of attempts

```
#define CORRECT_PASSWORD 1234

int main(void) {
  int password;
  int attempts = 0;
  do{
    printf ("Enter your password: ");
    scanf ("%i", &password);
    attempts = attempts +1;
  }while ((password!= CORRECT_PASSWORD)&&(attempts<3));

  if (password== CORRECT_PASSWORD){
     printf ("Welcome!\n");
  }else {
      // if password wrong we necessarily have exceeded
      // the number of attempts
      printf ("Sorry, only 3 attempts are allowed"):
      printf (", access denied\n");
  }

return 0;
}
```

Complex condition
The block is executed **while** the PASSWORD is not correct **AND** the number of attempts is smaller than 3

Post-check
The *while* loop may end because the password is found or because the attempts limit is reached.
We need to test after the loop which of the exit conditions holds

Integers from 50 to 1 or user exit

- Programa that prints to the screen integer number from 50 to 1 in decreasing number . Stops when 1 is reached or when the user selects to exit (user is prompted after each number if he/she wants to exit)

```
int main(void)
{
    int x = 50;   // x stores the integer value to display
    int continue;// continue stores user answer to continue prompt
    do{
       printf("%i \n", x); // display x
       //decrease x value
       x--; //an alternative way of writing x = x - 1;
       printf("Do you want to display the next number(YES->1; NO->0)? ");
       scanf ("%i", & continue);
    } while ((x>0) && (continue ==1));

    system("PAUSE");
    return 0;
}
```

CONTROL STRUCTURE NESTING

Control structure nesting

- nesting: enclosing control structures one into another
  - The instruction block of any structure can contain other structures

- **Days of the week**
- Program that prompts the user to enter a number and outputs the weekday corresponding to the number
- This will be repeated until the user decides to exit

switch structure nested inside a *do-while* loop.

```
int main(void){
  int continue;
  int n;

  do {
      printf("\n Enter an integer number [1..7]: ");
      scanf("%i", &n);
    switch (n) {
      case 1: printf(" Monday\n "); break;
      case 2: printf(" Tuesday\n "); break;
      case 3: printf(" Wednesday\n "); break;
      case 4: printf(" Thursday\n "); break;
      case 5: printf(" Friday\n "); break;
      case 6: printf(" Saturday\n "); break;
      case 7: printf(" Sunday\n "); break;
      default: printf(" Wrong number\n");
    }
      printf(" Do you want to continue? 1/0: ");
      scanf("%i", & continue);
  } while (continue==1) ;

  system("PAUSE");
  return 0;
}
```

- Write a Program that prompts the user to enter a number and outputs all integers from 1 to that number. Repeat until user wants to exit, this will be specified by entering a 1

for loop nested in a *do-while* loop

```
int main(void){
      int num, i;
      int salir;

    do {//Se repite hasta que el usuario inserte 1
        printf("Introduzca un numero");
        scanf("%d", &num);
        printf("Los numeros del 1 al %d son: ", num);
        for (i=1; i<=num; i++){
            printf("%d, ", i);
        }
        printf("\nDesea salir? (1-si, 0-no) ");
        scanf("%d", &salir);
    } while (salir!=1);

    system("PAUSE");
    return 0;
}
```

C programming - summary

- Structure of a C program

```
#include <stdio.h>
int main(void)
{
    variable declaration instructions...
     executable instructions ...
    return 0;
    }
```

- Assignment operator

```
=
```

- Variable declaration

```
datatype variable_name
const datatype constant_name = value;
```

- Input and output (for integers)

```
scanf("%i",&variable);
printf("%i", variable);
```

Conditional control structures

```
if (logical_expression_1){
        instruction_block_1;
}
else {
    if(logical_expression_2){
        instruction_block_2;
    }
    else {
        logical_expression_3;
    }
}
```

```
switch(selector){
    case value_1:
       instruction_block_1;
       break;
    case value_2:
       instruction_block_2;
       break;
    case value_n:
       instruction_block_n;
       break;
    default:
       instruction_block;
}
```

Iterative control structures (loops)

```
for (initialization; logical_expression; update) {

        instruction_block
}
```

```
while (logical_expression) {

 instruction_block

}
```

0 or more times

```
do {
        instruction_block

} while (logical_expression);
```

At least once

TEMA 4.
ESTRUCTURAS DE CONTROL

Universidad
Carlos III de Madrid
www.uc3m.es