Chapter 5

# Control Structures

Dr. Niroj Kumar Pani

*nirojpani@gmail.com*

Department of Computer Science Engineering & Applications
Indira Gandhi Institute of Technology
Sarang, Odisha

# Chapter Outline…

- Introduction
- Decision Control Structure
  - The if Statement
  - The switch Statement (Multi Way Decision)
  - Assignments - I
- Loop Control Structure
  - Introduction
  - The while Loop / Statement
  - The do-while Loop / Statement
  - The for Loop / Statement
  - Nesting of Loops

- **Jump Control Structure (Unconditional Jump)**
  - **Introduction**
  - **The** continue **Statement**
  - **The** break, return, **and** exit() **Statements**
  - **The** goto **Statement**
  - **Assignments - II**

# Introduction

- Control structures defines the order in which the statements in a program are executed.
- There are four types of control structures in C:
  1. Sequence control structure:
     - Control flows in a straight line.
     - All the statements in the program are executed one after another, starting from the 1st statement to the last statement.
     - All the programs that we have done so far, fall into this category.
  2. Selection / Decision / Branch control structure:
     - A condition is examined.
     - A statement (or a group of statements) is executed if the condition is true and another statement (or group of statements) is executed if the condition becomes false.

- **Implemented though the following statements / constructs:**
  - `if` **statement. It has four different forms:**
    - **Simple** `if` **statement**
    - `if...else` **statement**
    - **Nested** `if...else` **statement**
    - `else if` **ladder**
  - `switch` **statement**

## 3. Loop / Repetition / Iteration control structure:

- **A condition is examined.**
- **A statement (or a group of statements) is repeatedly executed till the condition is true.**
- **Implemented though the following statements / constructs:**
  - `while` **statement.**
  - `do...while` **statement**
  - `for` **statement**

## 4. Jump control structure (Unconditional jump):

- We jump from one statement to another unconditionally.
- Implemented though the following statements / constructs:
  - `continue` statement
  - `break` statement
  - `return` statement
  - `exit()` statement
  - `goto` statement.

*In the following sections, we shall discuss each of the control structures (except the sequence control structure) in detail.*

# Selection / Decision / Branch Control Structure

# The `if` Statement

## *Simple `if` (One-way decision)*

- **Syntax:**

```
                    if(expression)
                    {
   if block                statement(s);
                    }
```

Here,

– `expression` corresponds to a C expression, that is evaluable to either *true (non-zero)* or *false (zero).*

– `statement(s)` corresponds to a *set* of statements (0, 1, or more no. of statements).

- **Control Flow:**

  - **The `expression` is evaluated 1st .**
  - **If it is true (non-zero), the set of statements within the *if block* is executed (sequentially). The control is then transferred to statement immediately after the *if block* .**
  - **If it is false (zero), the set of statements within the *if block* are skipped and the control is directly transferred to the statement immediately after the *if block* .**

## Programming Example:

```
/* PR5_1.c: Input four integers a, b, c, d and print the value of (a+b)/(c-d) if (c-d) ≠0 */

# include <stdio.h>
# include <conio.h>

void main()
{
   int a, b, c, d;
   float result;
   printf("Enter four integers: ");
   scanf("%d %d %d %d", &a, &b, &c, &d);

   if((c-d)!=0)
   {
       result = (float)(a+b)/(c-d);
       printf("The value of (a+b)/(c-d) is: %f", result);
   }

   getch();
}
```

**Output**

*Run 1:*

```
Enter four integers: 12 3 8 3
The value of (a+b)/(c-d) is: 3.000000
```

*Run 2:*

```
Enter four integers: 12 3 8 8
```

- **Notes:**

1. **When the *if block* contains just a single statement then the curly braces become optional. i.e.,**

```
if(expression)
{
    statement 1;
}
statement 2;
```

is same as

```
if(expression)
    statement 1;
statement 2;
```

**However, this programming style should be avoided.**

2. **Never put a semicolon (;) after the *if clause* , because it will be same as writing no statements within the *if block.***

```
if(expression);
{
    statement 1;
}
statement 2;
```

is same as

```
if(expression)
    ;  /*A blank statement*/
statement 1;
statement 2;
```

3. **Suppose, in the `expression` your intension is to write x == 0, but by mistake you have written x = 0. i.e.,**

```
if(x==0)
{
    statement 1;
    statement 2;
}
```

is by mistake written as

```
if(x=0)
{
    statement 1;
    statement 2;
}
```

***Then what will happen?***

**In this case, the statements within the *if block* are never executed, no matter whether x is equals to zero or not. Because, in C, 0 is always false. When we write x=0, x becomes false i.e., the total expression becomes false.**

**So, be careful while writing x==0.**

## *The `if...else` Statement (Two-way decision)*

- **The *simple* `if` statement does nothing when the `expression` becomes false. If we want to execute one set of statements when the condition (`expression`) becomes true and some other set of statements when the condition (`expression`) becomes false , then we should use the `if...else` statement.**
- **Syntax:**

```
                if(expression)
                {
if block                statement(s);
                }
                else
                {
else block              statement(s);
                }
```

- **Control Flow:**
  - **The `expression` is evaluated 1st .**
  - **If it is true (non-zero), the set of statements within the *if block* is executed (sequentially). If it is false (zero), the set of statements within the *else block* is executed (sequentially).**
  - **After the execution of the set of statements either in the *if block* or in the *else block*, the control is transferred to the statement immediately after the `if...else` construct** (i.e., after the closing curly brace of the *else block* ).
- **Notes:**
  1. **When the *else block* contains just a single statement then the curly braces becomes optional. i.e.,**

```
else
{
    statement 1;
}
statement 2;
```

is same as

```
else
    statement 1;
statement 2;
```

**However, this programming style should be avoided.**

## Programming Example 1:

```
/* PR5_2.c: The same program as in PR5_1, but in this case, it will print "(c-d) = 0, The
result is undetermined.", if (c-d) becomes equal to 0 */

# include <stdio.h>
# include <conio.h>

void main()
{
   int a, b, c, d;
   float result;

   printf("Enter four integers: ");
   scanf("%d %d %d %d", &a, &b, &c, &d);
   if((c-d)!=0)
   {
       result = (float)(a+b)/(c-d);
       printf("The value of (a+b)/(c-d) is: %f", result);
   }
   else
   {
       printf("(c-d) = 0, The result is undetermined.");
   }
   getch();
}
```

**Output**

*Run 1:*

```
Enter four integers: 12 3 8 3
The value of (a+b)/(c-d) is: 3.000000
```

*Run 2:*

```
Enter four integers: 12 3 8 8
(c-d) = 0, The result is undetermined.
```

## Programming Example 2:

```
/* PR5_3.c: Program to enter an integer and check whether it is even or odd*/

# include <stdio.h>
# include <conio.h>

void main()
{
   int a;
   printf("Enter an integer: ");
   scanf("%d", &a);
   if(a%2 == 0)
   {
       printf("\n%d is an even integer.", a);
   }
   else
   {
      printf("\n%d is an odd integer.", a);
   }

   getch();
}
```

**Output**

*Run 1:*

```
Enter an integer: 12
12 is an even integer.
```

*Run 2:*

```
Enter an integer: 5
5 is an even integer.
```

# *The Nested `if...else` Statement (Multi way decision)*

- **When we put an entire** `if` **statement or an entire** `if...else` **statement within the** *if block* **or the** *else block* **or** *both* **of another** `if...else` **statement, then the construct is called** nested `if...else`.

- **Syntaxes:**

```
if(expression1)
{
    statement(s);
    if(expression2)
    {
        statement(s);
    }
    else
    {
        statement(s);
    }
    statement(s);
}
```

```
if(expression1)
{
    if(expression2)
    {
        statement(s);
    }
}
else
{
    if(expression3)
    {
        statement(s);
    }
    else
    {
        statement(s);
    }
    statement(s);
}
```

- **Programming Example:**

```
/* PR5_4.c: Program to find the largest among three integers*/

# include <stdio.h>
# include <conio.h>

void main()
{
   int a, b, c, largest;
   printf("Enter three integers: ");
   scanf("%d %d %d", &a, &b, &c);
   if(a>b)
   {
       if (a>c)
           largest = a;
       else
           largest = c;
   }
   else
   {
      if (c>b)
           largest = c;
      else
           largest = b;
   }
```

*[Cont.]*

```
    printf("\nThe largest number is: %d", largest);
    getch();
 }
```

**Output**

```
Enter three integers: 12 15 2
The largest number is: 15
```

- **Notes:**

  1. **Dangling Else Problem:** **While nesting, care should be exercised to match every `else` statement with an `if` statement. When an `else` statement has no matching `if`, then that `else` is called dangling `else`.**

```
if()
    if()
        if()
        else
    else
else
else  /* The dangling else*/
```

2. **How to know which `else` belongs to (matches to) which `if`?:**

**The answer is simple. An `else` is always matched with the nearest unmatched `if`.**

```
1     if()
2     if()
3     if()
4     else  /*Belongs to the if() in line no. 3*/
5     else  /*Belongs to the if() in line no. 2*/
6     else  /*Belongs to the if() in line no. 1*/
7     if()
8     else  /*Belongs to the if() in line no. 7*/
9     if()
10    else  /*Belongs to the if() in line no. 9*/
11    else  /*Dangling else*/
```

3. **Try to avoid nested `if...else` statements unless until it is highly required.** **There are two reasons for it:**

   i. **Nested `if...else` statements are complex to understand and handle.**

   ii. **Almost all the programs that requires nested `if...else` construct, can also be easily done with the `else if` ladder.**

   *We will discuss the `else if` ladder next. We will see that the program (PR5_4.c) that we have done by using the `if...else` construct, can be done very easily with the `else if` ladder .*

## *The `else if` Ladder (Multi-way decision)*

- **The `else if` ladder does the same thing as that of the nested `if...else` construct** (both are meant for multi way decision making)**, but in a simpler manner.**
- **Syntax:**

```
if(expression1)
{
   statement(s);
}
else if(expression2)
{
   statement(s);
}
else if(expression3)
{
   statement(s);
}
else
{
   statement(s);
}
```

The total structure contains only one `if` at the beginning, and only one `else` at the end.

- **Control Flow:**
  - **The `expressions` are evaluated from the top (of the ladder), downwards.**
  - **As soon as an `expression` is found to be true, the block of statements associated with that `expression` is executed** (no other statement block is executed).
  - **If all the `expressions` are evaluated to be false, then the final `else` statement is executed.**
  - **After the execution of any one block of statements present in the ladder (either the `if` block, or any one of the `else if` blocks, or the final `else` block), the control is transferred to the statement immediately after `else if` ladder (skipping the rest of the ladder).**

## ■ Programming Example 1:

```
/* PR5_5.c: The same program as that of PR5_4.c (Program to find the highest among
three integers), using else if ladder*/

# include <stdio.h>
# include <conio.h>

void main()
{
   int a, b, c, greatest;
   printf("Enter three integers: ");
   scanf("%d %d %d", &a, &b, &c);

   if(a>b && a>c)
       greatest = a;
   else if(b>a && b>c)
       greatest = b;
   else
       greatest = c;

   printf("\nThe greatest number is: %d", greatest);

   getch();
}
```

**Output**

```
Enter three integers: 12 15 2
The greatest number is: 15
```

- **Programming Example 2:**

An electric power distribution company charges its domestic consumers as follows:

| Consumption Units | Rate of Charge |
|---|---|
| 0 - 200 | Rs. 0.50 per unit |
| 201 - 400 | Rs. 100 plus Rs.0.65 per unit excess of 200 |
| 401 - 600 | Rs. 230 plus Rs.0.80 per unit excess of 400 |
| 601 and above | Rs. 390 plus Rs.1.00 per unit excess of 600 |

Write a program that reads the customer number and power consumed and prints the amount to be paid by the customer.

/* PR5_6.c: Electricity Bill Estimation */

**Output**

```
Enter CUSTOMER NO. & UNITS consumed: 11 340
Customer No: 11, Charges = 358.00
```

```
# include <stdio.h>
# include <conio.h>

void main()
   {
       int  units, custNo;
       float chrges;

       printf("Enter CUSTOMER NO. & UNITS consumed: ");
       scanf("%d %d", &custNo, &units);

       if (units <= 200)
          chrges = 0.5 * units;
       else if (units <= 400)
          chrges = 100 + 0.65 * (units - 200);
       else if (units <= 600)
          chrges = 230 + 0.8 * (units - 400);
       else
          chrges = 390 + (units - 600);

       printf("\nCustomer No: %d, Charges = %.2f\n", custNo, chrges);
       getch();
   }
```

# *[NOTE]: if...else Vs. The Conditional Operator (? :)*

- **We have studied the conditional operator (? : ) in the chapter “Operators and Expressions”. The working of conditional operator is exactly same as that of the `if...else` construct. i.e.,**

```
(expression1) ? (expression2) : (expression3);
```

**is same as**

```
if (expression1)
{
    expression2;
}
else
{
    expression3;
}
```

**Then what is the need of the `if...else` statements?**

- **The limitation of the conditional operator is that, after the ? or after the : only one C statement can be written.**

# The switch Statement (Multi Way Decision)

- **It is rather called the** switch-case-break **construct.**
- **Syntax:**

```
switch(expression)
{
   case constant1:
         statement(s);
         break;
   case constant2:
         statement(s);
         break;

   ...
   ...

   default:
         statement(s);
}
```

Here,

- expression corresponds to either an integer/character constant like 1, 2, 3, ‘a’, ‘b’, ‘c’ etc., or any C expression that is evaluable to an integer/character value.
- constant1, constant2,... are integer/character constants like 1, 2, 3, ‘a’, ‘b’, ‘c’ etc. Each of these constants should be *unique* within a switch-case construct.
- The break statements are optional.
- The default level is optional. There can be at most one default level.
- The default level may be placed any where but usually placed at the end.

- **Control Flow:**
  - **The** `expression` **is evaluated 1st.**
  - **Its value is then matched, one by one, in order, against** `constant1, constant2, . . .` **that follow the** `case` **levels. When a match is found, the program executes the set of statements corresponding to *that*** `case` **and all subsequent** `case` **and** `default` **as well (if a** `default` **is present). The control is then transferred to the statement immediately after the** switch-case **construct** (i.e., after the closing curly brace of the switch-case construct).

    **However, the** `break` **statement is meant for taking the control out of the current block (a block is a set of statements enclosed within a pair of curly braces).**

    **So, when a** `break` **statement is present within a** `case`**, it takes the control out of the** switch-case **construct on execution** (i.e., transfers the control to the statement immediately after the switch-case construct). **So, in this situation the set of statements corresponding to *only that*** `case` **is executed.**

- **If no match is found with any of the** `case` **(that precedes the** `default`**), the program executes the set of statements corresponding to the** `default` **and all subsequent** `case` (if at all present after the `default` ; recall that `default` can be placed any where). **The control is then transferred to the statement immediately after the** switch-case **construct.**

- **Examples: A few examples will clarify the control flow**

<table>
<tr><th>Sl. No.</th><th>Example</th><th>Output</th></tr>
<tr><td>1</td><td>

```c
void main()
{
    int i = 2;

    switch(i)
    {
        case 1:
            printf("I am in case 1 \n");
        case 2:
            printf("I am in case 2 \n");
        case 3:
            printf("I am in case 3 \n");
        default:
            printf("I am in default \n");
    }
}
```

</td><td>

```
I am in case 2
I am in case 3
I am in default
```

</td></tr>
</table>

<table>
<tr><th>Sl. No.</th><th>Example</th><th>Output</th></tr>
<tr><td>2</td><td><pre>
void main()
{
    int i = 2;

    switch(i)
    {
        case 1:
            printf("I am in case 1 \n");
        case 2:
            printf("I am in case 2 \n");
        case 3:
            printf("I am in case 3 \n");
            break;
        default:
            printf("I am in default \n");
    }
}
</pre></td><td><pre>
I am in case 2
I am in case 3
</pre></td></tr>
</table>

| Sl. No. | Example | Output |
|---|---|---|
| 3 | `void main()`<br>`{`<br>`    int i = 2;`<br><br>`    switch(i)`<br>`    {`<br>`        case 1:`<br>`            printf("I am in case 1 \n");`<br>`        case 2:`<br>`            printf("I am in case 2 \n");`<br>`            break;`<br>`        case 3:`<br>`            printf("I am in case 3 \n");`<br>`        default:`<br>`            printf("I am in default \n");`<br>`    }`<br>`}` | `I am in case 2` |

<table>
<tr><th>Sl. No.</th><th>Example</th><th>Output</th></tr>
<tr><td>4</td><td><pre>
void main()
{
    int i = 2;

    switch(i)
    {
        case 1:
            printf("I am in case 1 \n");
            break;
        case 2:
            printf("I am in case 2 \n");
            break;
        case 3:
            printf("I am in case 3 \n");
            break;
        default:
            printf("I am in default \n");
    }
}
</pre></td><td><pre>I am in case 2</pre></td></tr>
</table>

<table>
<tr><th>Sl. No.</th><th>Example</th><th>Output</th></tr>
<tr><td>5</td><td><pre>
void main()
{
    int i = 10;

    switch(i)
    {
        case 1:
            printf("I am in case 1 \n");
            break;
        case 2:
            printf("I am in case 2 \n");
            break;
        case 3:
            printf("I am in case 3 \n");
            break;
        default:
            printf("I am in default \n");
    }
}
</pre></td><td><pre>
I am in default
</pre></td></tr>
</table>

<table>
<tr><th>Sl. No.</th><th>Example</th><th>Output</th></tr>
<tr><td>6</td><td>

```c
void main()
{
    int i = 10;

    switch(i)
    {
        case 1:
            printf("I am in case 1 \n");
            break;
        default:
            printf("I am in default \n");
        case 2:
            printf("I am in case 2 \n");
            break;
        case 3:
            printf("I am in case 3 \n");
            break;
    }
}
```

</td><td>

```
I am in default
I am in case 2
```

</td></tr>
</table>

| Sl. No. | Example | Output |
|---|---|---|
| 7 | void main()<br>{<br>    int i = 10;<br><br>    switch(i)<br>    {<br>        case 1:<br>            printf("I am in case 1 \n");<br>            break;<br>        default:<br>            printf("I am in default \n");<br>            break;<br>        case 2:<br>            printf("I am in case 2 \n");<br>            break;<br>        case 3:<br>            printf("I am in case 3 \n");<br>            break;<br>    }<br>} | I am in default |

| Sl. No. | Example | Output |
|---|---|---|
| 8 | void main()<br>{<br>    char ch = 'B';<br><br>    switch(ch)<br>    {<br>        case 'a':<br>        case 'A':<br>            printf("'A' for apple \n");<br>            break;<br>        case 'b':<br>        case 'B':<br>            printf("'B' for ball \n");<br>            break;<br>        case 'c':<br>        case 'C':<br>            printf("'C' for cat \n");<br>            break;<br>    }<br>} | ‘B’ for ball |

## Programming Example 1:

```
/* PR5_7.c: Program that reads an alphabet and prints whether it is a vowel or consonant*/

# include <stdio.h>
# include <conio.h>

void main()
{
    char ch;
    printf("Enter an alphabet: ");
    scanf("%c", &ch);
    switch(ch)
    {
        case 'a':
        case 'A':
        case 'e':
        case 'E':
        case 'i':
        case 'I':
        case 'o':
        case 'O':
        case 'u':
        case 'U':
            printf("\nIt is a vowel.\n");
            break;
```

*[Cont.]*

```
        default:
            printf("\nIt is a consonant.\n");
    }
}
```

**Output**

```
Run 1:
Enter an alphabet: p
It is a consonant.

Run 2:
Enter an alphabet: u
It is a vowel.
```

- **Programming Example 2:**

For a student’s mark within 0-100, the index = (mark/25). The grades are calculated as follows:

| Index | Grade |
|---|---|
| 0 | D |
| 1 | C |
| 2 | B |
| 3 | A |

Write a program that reads the mark within 0-100 and prints the appropriate grade.

```
/* PR5_8.c: Student’s Grade Calculation */

# include <stdio.h>
# include <conio.h>

void main()
{
    int mark, index;
    printf("Enter mark (0-100): ");
    scanf("%d", &mark);
    index = mark/25;

    switch(index)
    {
        case 0:
            printf("\nThe grade is: D\n");
            break;
        case 1:
            printf("\nThe grade is: C\n");
            break;
        case 2:
            printf("\nThe grade is: B\n");
            break;
        case 3:
            printf("\nThe grade is: A\n");
            break;
```

*[Cont.]*

```
        default:
            printf("\nYou haven’t entered the mark within 0-100.");
    }
}
```

## Output

*Run 1:*

```
Enter mark (0-100): 85
The grade is: A
```

*Run 2:*

```
Enter mark (0-100): 145
You haven’t entered the mark within 0-100.
```

- **Notes:**

  1. **Though in all our programs we have put the `case` in some order, one can put the `case` in any order he likes. But the matching is always done in top to bottom order.**

```
void main()
{
    int i = 10;
    switch(i)
    {
        case 78:
            printf("I am in case 78 \n");
            break;
        case 10:
            printf("I am in case 10 \n");
            break;
        case 196:
            printf("I am in case 196 \n");
            break;
        default:
            printf("I am in default \n");
    }
}
```

2. **One can mix characters and integers in the `cases`** (characters are actually integers).

3. **Every statement in a switch-case must belong to some `case`. If a statement doesn’t belong to some case the compiler WON’T report an error, but the statement would never be executed.**

```
...
switch(i)
{
   printf(“Enter a value: ”); /* This statement is never executed*/
   case 100:
       j = i+50;
       printf(“%d \n“, j);
       break;
   case 200:
       j = i-50;
       printf(“%d \n“, j);
       break;
}
...
```

4. Though, the `default` level may be placed any where it should always be placed at the end.
5. In principle, a switch-case construct can be nested, but it is rarely practiced.
6. The switch-case construct is very helpful in writing menu driven programs.

## *Switch-case Vs. The else-if Ladder*

- Though both are meant for multi way decision making, there are some thing that simply can't be done by using the switch-case:
  1. switch-case are meant for equality comparisons. One can't write a case that looks like: `case i<=100`.
  2. A float value (any value other than integer/character) can't be tested by using a switch .
  3. A `case` can't contain an expression like, `a+3`.
  4. Multiple `case` can't use the same expression.

# Assignments - I

**Complete the experiments given in** “Lab Manual - Section 4”**.**

# Loop / Repetition / Iteration Control Structure

# Introduction

- What is Looping?: Executing a set of statements repeatedly till a particular condition is true.
  - A *condition* is nothing but an *expression,* evaluable to either true (non-zero) or false (zero).
  - The condition *tests* a variable, known as the *control variable* that controls the number of times the loop is executed.
- The Overall Looping Process: The looping process, in general, involves the following four steps
  1. Initialization: The *control variable* is assigned to some initial value.
  2. Testing Using a Condition: The *control variable* is tested (using an expression). The result is either true (non-zero) or false (zero).
  3. Executing the set of statements in the body of the loop.
  4. Update: The *control variable* is updated (incremented, decremented, or any other operation that changes the value of the *control variable*).

- **Classification:** **Depending upon the position of the condition, a loop construct may be classified as**
  1. **Entry controlled loop (pre-test loop)**
     - `while` **loop**
     - `for` **loop**
  2. **Exit controlled loop (post-test loop)**
     - `do-while` **loop**

# The while Loop / Statement

- **Syntax:**

```
                 initialization;
                 ...
                 while(condition)
                 {
   body of           statement(s);
   the loop          update;
                 }
```

- **Control Flow: The `while` is an entry-controlled loop.**
  - **The `condition` is evaluated 1st. If it is true (non-zero), then the body of the loop is executed. After the execution of the body, the `condition` is once again evaluated and if it is true, the body is executed once again. This process is repeated until the `condition` finally becomes false (zero).**
  - **When the `condition` becomes false, the loop is terminated, and the control goes to the statement immediately after the body of the loop.**

## ■ Programming Example 1:

/* PR5_9.c: A program to calculate the sum of squares of numbers between 1 to 10. i.e., sum = $1^2+2^2+3^2+$ ..... $+10^2$*/

```
# include <stdio.h>
# include <conio.h>

void main()
{
    int sum = 0;
    int i = 1;                  /*Initialization*/

    while(i<=10)                /*Condition (Testing)*/
    {
        sum = sum + (i*i);
        i++;                    /*Update (Incrementing)*/
    }

    printf("The sum is: %d\n\n", sum);
}
```

## Output

```
The sum is: 385
```

## ■ Programming Example 2:

/* PR5_10.c: A program to evaluate the equation $y = x^n$ when n is a non-negative integer */

```
 # include <stdio.h>
 # include <conio.h>

 void main()
 {
     int count, n;
     float x, y;

     printf("Enter the values of x and n: ");
     scanf("%f %d", &x, &n);
     y = 1.0;
     count = 1;              /* Initialization */

     while (count <= n)      /*Condition (Testing)*/
     {
         y = y*x;
         count++;            /* Update (Incrementing) */
     }

     printf("\nx = %f; n = %d; \nx to power n = %f\n",x,n,y);
 }
```

**Output**

```
Enter the values of x and n: 12 5
x = 12.000000; n = 5;
x to the power n = 248832.000000
```

- **Notes:**

1. **If you forget to update the *control variable*** (i.e., forget to write `i++` or `count++`, as written in the last two programs within the body of the while loop), **the loop becomes an infinite loop.**

2. **It is not necessary that the *control variable* must only be an `int`. It could also be a `float` (any numeric value).**

   **Again, the update doesn't mean only incrementing or decrementing. It could be any operation, that eventually changes the *control variable,* so that the condition becomes false at some time.**

```
void main()
{
     float a = 10.0;
     while(a <= 1000.0)
     {
          printf("Hi\n"); /* "Hi" is printed 6 times*/
          a = a * 2.5;
     }
}
```

3. **Never put a semicolon (;) immediately after the *while clause*. It will lead to an infinite loop.**

```
int i;
...
while(i <= 10);
{
    printf("%d\n", i);
    i++;
}
```

is same as

```
int i;
...
while(i <= 10)
        ;
{
    printf("%d\n", i);
    i++;
}
```

4. **What do you think would be the out put of the following program?**

```
int i;
...
while(i = 10)
{
    printf("%d\n", i);
    i++;
}
```

**It is an infinite loop, because the condition `(i = 10)` is always true (non-zero).**

5. **What do you think would be the out put of the following program?**

```
void main()
{
    int i = 1;
    while(i <= 32767)
    {
        printf(“%d\n“, i);
        i++;
    }
}
```

**No, it doesn’t print numbers from** 1 to 32767. **It is an infinite loop.**

**To begin with, it prints out numbers from** 1 to 32767. **After that, the value of** `i` **is incremented to** 1, **therefore it tries to become** 32768, **which falls outside the valid integer range** (assuming that, the compiler gives 2 bytes for an `int`), **so it goes to the other side and becomes** -32768, **which again satisfies the condition** `(i <= 32767)`**. This process goes on indefinitely.**

# The `do-while` Loop / Statement

- **The Need (The difference between while and do-while): The `while` loop is an *entry-controlled loop* - meaning that - it executes the body of the loop only if the condition is true.**

  **However, on some occasions it might be necessary to execute the body of a loop at least once, even if the condition becomes false. In such situations, we should use an *exit-controlled* loop, like the `do-while` loop.**

- **Syntax:**

```
initialization;
...
do
{
    statement(s);
    update;
} while(condition);
```

body of the loop

*Notice the semicolon (;). It was not present in the syntax of the while loop.*

- **Control Flow:** **As already mentioned the** `do-while` **is an exit-controlled loop.**
  - **On reaching the** `do` **statement, the control proceeds to execute the body of the loop first.**
  - **At the end of the loop, the** `condition` **in the** `while` **statement is evaluated. If it is true (non-zero), then the body of the loop is executed once again. This process is repeated till the** `condition` **finally becomes false (zero).**
  - **Eventually, when the** `condition` **becomes false, the loop is terminated, and the control is transferred to the statement immediately after the** `while` **statement.**

## Programming Example:

```
/* PR5_11.c: Program that continues to read a number and displays its square until the use says
“NO”*/

# include <stdio.h>
# include <conio.h>

void main()
{
    char status = ‘Y’; /*Initialization*/
    int n;
    do
    {
        printf("\n\nEnter an integer: ");
        scanf("%d", &n);
        printf("\nIts square is: %d", (n*n));
        printf("\n\nWould you like to continue (Y/N)?: ");
        status = getche(); /*Update*/
    }while(status == 'Y' || status == 'y'); /*Condition (Testing)*/
}
```

```
Enter an integer: 5
Its square is: 25
Would you like to continue (Y/N)?: Y
...
```

**Output**

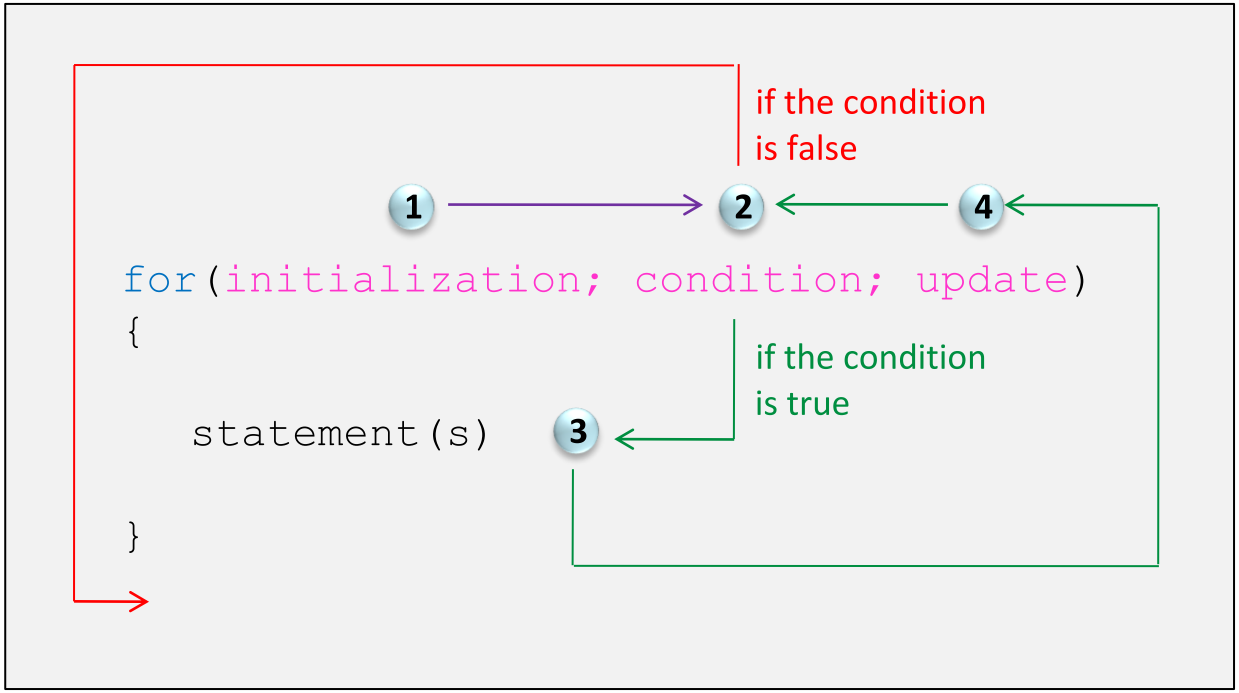
# The `for` Loop / Statement

- The `for` loop is another entry-controlled loop that provides a more concise loop control structure. It allows initialization, testing using a condition, and update in a single line.
- Syntax:

```
for(initialization; condition; update)
{
    statement(s);
}
```

body of
the loop

- **Control Flow:** **Demonstrated thorough the following diagram**



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

    statement(s)


}
```

## ■ Programming Example 1:

```
/* PR5_12.c The same program as that of PR5_9.c using the for loop
 (A program to calculate sum = $1^2+2^2+3^2+ ..... +10^2$) */

# include <stdio.h>
# include <conio.h>

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

    for(i=1; i<=10; i++)  /*Initialization, Condition (Testing), and Update*/
    {
        sum = sum + (i*i);
    }

    printf("The sum is: %d\n\n", sum);
}
```

## Output

```
The sum is: 385
```

## ■ Programming Example 2:

```c
/* PR5_13.c A program to calculate the nth Fibonacci number */
/* Note: The Fibonacci series is : 0 1 1 2 3 5 8 ... */

# include <stdio.h>
# include <conio.h>

void main()
{
    int n, i, fib1=0, fib2=1, fib;

    printf("\n\nEnter the value of n: ");
    scanf("%d", &n);

    for(i=1; i<=n-2; i++)
    {
       fib = fib1+fib2;
       fib1 = fib2;
       fib2 = fib;
    }

    printf("\nThe %dth Fibonacci number is: %d", n, fib);
    getch();
}
```

**Output**

```
Enter the value of n: 7
The 7th Fibonacci number is: 8
```

- **Notes:**

1. **In a `for` loop, both the *initialization* and the *update* sections can contain more than one expressions. If done so, the expressions should be separated by commas (,). For example:**

```
j=1;
for(i=0; i<10; i++)
{
    ...
}
```

*Can be rewritten as*

```
/* More than one expressions initialized*/
for(i=0, j=1; i<10; i++)
{
    ...
}
```

```
for(i=0; j<10; i++)
{
    ...
    j++;
}
```

*Can be rewritten as*

```
/* More than one expressions updated*/
for(i=0; j<10; i++, j++)
{
    ...
}
```

```
j=1;
for(i=0; j<10; i++)
{
    ...
    j++;
}
```

*Can be rewritten as*

```
/* More than one expressions initialized
   and updated*/
for(i=0, j=1; j<10; i++, j++)
{
    ...
}
```

2. However, the *condition* section must contain *exactly one* expression (the expression may contain only the *control variable*, or other variables along with the control variable).

3. Writing the *initialization,* or the *condition,* or the *update* sections, within the `for` statement, is optional. However, the semicolons (;) separating the sections must remain.

   The following examples will clarify the concept

```
...
i=5;  /* Initialization is written here */

for( ; i<100 ; i=i+5)  /* Contains the condition and the update*/
{
   printf("%d\n", i);
}
...
```

```
...
i=5;  /* Initialization is written here */

for( ; i<100 ; )  /* Contains only the condition */
{
   printf(“%d\n”, i);
   i = i+5;  /* Update is written here */
}
...
```

```
...
i=5;  /* Initialization is written here */

for( ; ; )  /* Contains only the semicolons*/
{
   printf(“%d\n”, i);
   if (i>100)  /* Condition is written here */
      break;
   i = i+5;  /* Update is written here */
}
...
```

4. **If we completely remove the *condition* section, or the *update* section, or both form the `for` loop then it will be an infinite loop.**

**The following examples will clarify the concept**

```
for(i=0;   ;  i++)  /* No condition; Infinite loop*/
{
   printf(“%d\n”, i);
}
```

```
for(i=0;  i<10  ;  )  /* No update; Infinite loop*/
{
   printf(“%d\n”, i);
}
```

```
for(i=0;   ;  )  /* No condition and update; Infinite loop*/
{
   printf(“%d\n”, i);
}
```

```
for(;  ;)  /* No condition and update; Infinite loop (the easiest way to write an
{                                                        infinite loop)*/
   printf(“%d\n”, i);
}
```

# Nesting of Loops

- **The way `if` and `switch` statements can be nested, similarly the loops can also be nested by placing one within another.**
- **Programming Example 1:**

```
/* PR5_14.c A program to display all prime numbers from 1 to n */

# include <stdio.h>
# include <conio.h>

void main()
{
   int i, j, n, num;

   printf("\nEnter a range: ");
   scanf("%d", &n);
   printf("\nThe prime numbers within the range 1-%d are: ", n);
```

*[Cont.]*

```
    for(i=1;i<=n;i++)
    {
        num = i;
        for(j=2;j<num;j++)
        {
            if(num%j == 0)
                break;
        }
        if(num == j)
        {
            printf("%d ", num);
        }
    }
    getch();
}
```

## Output

```
Enter a range: 20
The prime numbers within the range 1-20 are: 2 3 5 7 11 13 17 19
```

- **Programming Example 2: Write a program to print the following structure**

```
1
1  2
1  2  3
1  2  3  4
```

```
/* PR5_15.c A program to display right pyramid */

# include <stdio.h>
# include <conio.h>

void main()
{
   int row, col;

   for(row=1;row<=4;row++)
   {
       for(col=1;col<=row;col++)
       {
           printf("%d  ", col);
       }
       printf("\n\n");
   }
   getch();
}
```

- **Programming Example 3:** **Write a program to print the following structure**

```
            1
        1   2   1
    1   2   3   2   1
1   2   3   4   3   2   1
```

```
/* PR5_16.c A program to display full pyramid */

# include <stdio.h>
# include <conio.h>

void main()
{
   int row, col, space;

   for(row=1;row<=4;row++)
   {
       for(space=1;space<=4-row;space++)
           printf("   ");

       for(col=1;col<=row;col++)
           printf("%d  ", col);

       for(col=col-2;col>=1;col--)
           printf("%d  ", col);

       printf("\n\n");
   }
   getch();
}
```

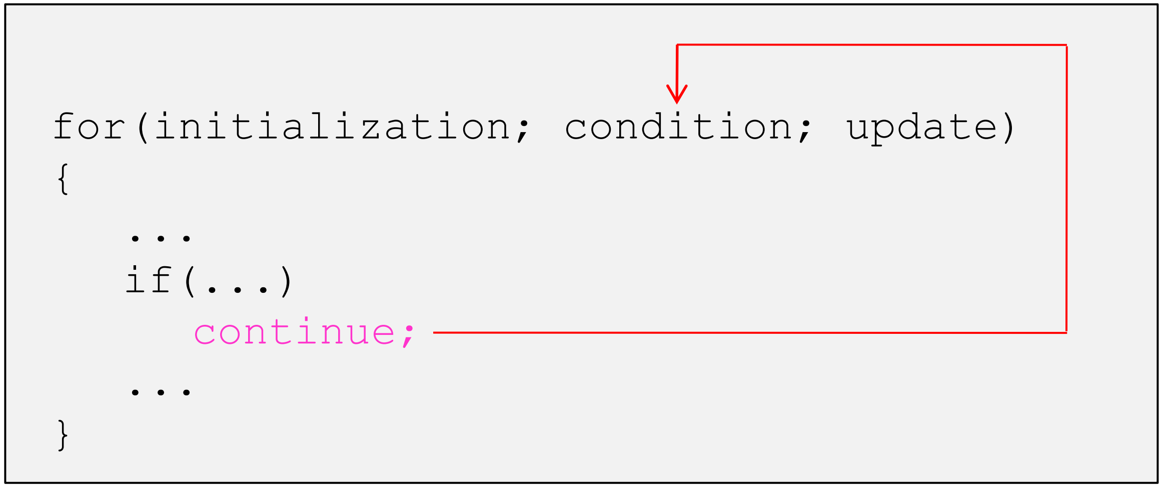
# Jump Control Structure

# Introduction

- **The jump (unconditional jump) is implemented through the following statements.**
  - `continue`
  - `break`
  - `return`
  - `exit()`
  - `goto`
- `exit()` **is a library function; rest are key words.**

# The `continue` Statement

- The `continue` statement can *only* be used within a loop construct.
- **Syntax:** `continue;`
- **What It Does?:** When executed (within a loop), it takes the control directly to the next iteration (i.e., to the *condition* clause) of the *current* loop, skipping all the statements after the `continue` statement within the loop.
  - A `continue` is usually associated with an `if`.



```
for(initialization; condition; update)
{
    ...
    if(...)
        continue;
    ...
}
```

## Programming Example:

```
/* PR5_17.c A program to display the odd numbers between 1 to 10 */

# include <stdio.h>
# include <conio.h>

void main()
{
   int i;
   printf("\nThe odd numbers between 1-10 are: ");
   for(i=1;i<=10;i++)
   {
       if(i%2 == 0)
       {
           continue;
       }
       printf("%d ", i);
   }
   getch();
}
```

**Output**

```
The odd numbers between 1-10 are: 1 3 5 7 9
```

# The `break`, `return`, and `exit()` Statements

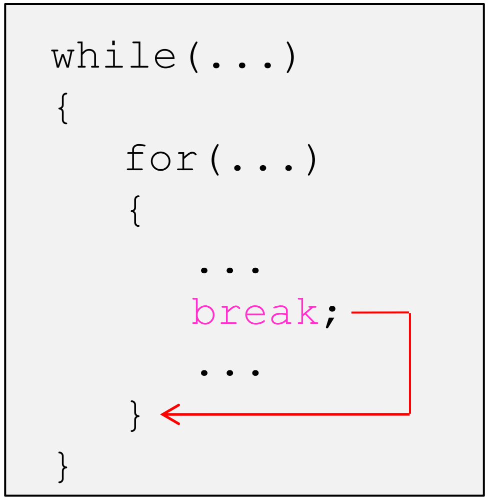
- **Unlike the `continue` statement which can be explicitly used within a loop, the `break`, `return`, and `exit()` statements can be used any where in a program (there is no restriction).**

### *break*

- **We have already used `break` statements before.**
- **Syntax:**

```
break;
```

- **What it Does?: When executed, it takes the control out of the current block** (recall that, a block is a set of statements enclosed within a pair of curly braces).



```
while(...)
{
    for(...)
    {
        ...
        break;
        ...
    }
}
```

## ***return***

- **Syntax:** `return [(][expression][)];`

  The components in the square brackets are optional.

- **What it Does?:** **The `return` statement terminates the execution of the current function and takes the control to the calling function immediately following the function call. A return statement can also return a value to the calling function.**

  *More on the `return` statement will be discussed in the chapter "Functions".*

## *`exit()`*

- **Syntax:** `exit([integer_constant]);`

  The component in the square brackets is optional.

- **What it Does?: The `exit()` statement (function) takes the control out of the whole program (i.e., terminates the program).**
  - **`exit()` optionally takes an integer constant as its argument. Normally, a zero as an argument (`exit(0)`) is used to indicate normal termination of the program (to the operating system) and a non-zero value as an argument is used to indicate termination of program due to some error or abnormal condition.**
  - **An `exit()` is usually associated with an `if`.**

- [NOTE]: The description of `exit()` is present in the header file “`stdlib.h`”. So, in order to use `exit()` we must include the header file “`stdlib.h`” in our program through the preprocessor directive **`#include<stdlib.h>`**, otherwise we may get a warning.

## Programming Example:

```c
/* PR5_18.c A program that tests a number to be prime or not (A prime number is a natural number
greater than 1 that has no positive divisors other than 1 and itself.) */

# include <stdio.h>
# include <conio.h>
# include <stdlib.h>

void main()
{
   int i, num;
   printf("\nEnter an positive integer: ");
   scanf("%d", &num);
   for(i=2;i<=num-1;i++)
   {
       if(num%i == 0)
       {
           printf("\n%d is a NOT a prime number.\n\n", num);
           exit(0);
       }
   }
   printf("\n%d is a prime number.\n\n", num);
}
```

**Output**

```
Enter a number: 56
56 is NOT a prime number.
```

# The goto Statement

- A word of caution: Avoid goto statement. It is not at all required for a highly structured language like C (we are discussing it for the shake of completeness). There are two reasons for it:
    1. It obscures the normal control flow of the program (as we shall see, a `goto` statement can cause the control to jump anywhere in the program without any reason). Hence, the program becomes difficult to understand and debug.
    2. Almost always, we can write the same program by using other control statements like `if, switch, exit()` etc., in an easy and more elegant manner.

- **Syntax:**

```
...
goto label:
...
label:
   statement;
...
```

(Forward Jump)

```
...
label:
   statement;
...
goto label:
...
```

(Backward Jump)

*Few Explanations*

- The `goto` requires a `label` in order to identify the place of jump. The `label` is nothing but an identifier name.
- The `label` must be followed by a colon.

- **Control Flow: During the execution of a program, when a statement like “`goto begin;`” is met, the control will jump to the statement immediately following the label “`begin:`”. This happens unconditionally.**

[NOTE]: In a backward jump (when the “`label:`” is placed before the “`goto label;`” statement), the program will fall in an infinite loop if no condition is specified (though another `goto` or an `if` statement) to take the control after the “`goto label;`” statement.

- **Programming Example 1** (This is the 1st and last time we are doing a program using goto):

```
/* PR5_19.c A program that illustrate the use of goto */

# include <stdio.h>
# include <conio.h>

void main()
{
   int i, j, k;
   for(i=1;i<=2;i++)
   {
       for(j=1;j<=2;j++)
       {
           for(k=1;k<=2;k++)
           {
              if(i==2 && j==2 && k==2)
                  goto end;
              else
                  printf("%d %d %d\n", i, j, k);
           }
       }
   }
   end:
       printf("\nOut of the loop at last !!\n\n");
}
```

**Output**

```
111
112
121
122
211
212
221

Out of the loop at last !!
```

## ■ Programming Example 2 (Rewriting the previous program without using goto):

```
/* PR5_20.c: Program to print the series as in PR5_19.c without using goto. */

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

void main()
{
   int i, j, k;
   for(i=1;i<=2;i++)
   {
       for(j=1;j<=2;j++)
       {
           for(k=1;k<=2;k++)
           {
              if(i==2 && j==2 && k==2)
              {
                 printf("\nOut of the loop at last!!\n\n");
                 exit(0);
              }
              else
                  printf("%d %d %d\n", i, j, k);
           }
       }
   }
}
```

**Output**

```
111
112
121
122
211
212
221

Out of the loop at last !!
```

# Assignments - II

**Complete the experiments given in “Lab Manual - Section 5”.**

# End of Chapter 5