CSE101-Lec#6

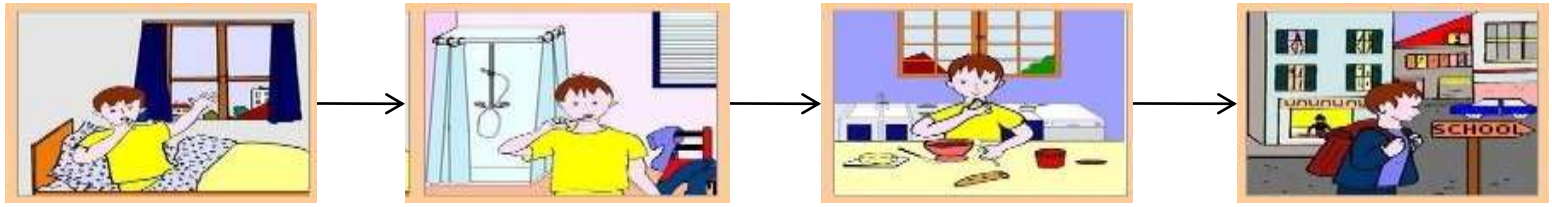
Control structures

Outline

- Control structure
  - Decision Statements
    - If statement
    - If-else statement
    - Switch statement

Program

- Program is a set of instruction executed one by one.



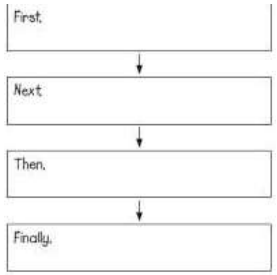
- Depending upon the circumstances sometimes it is desirable to alter the sequence of execution of statements.

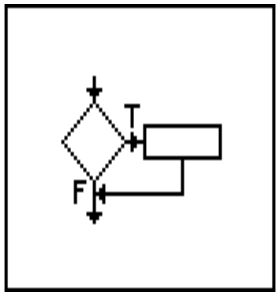
Control Statements

- The C language programs until now follows a sequential form of execution of statements.
- C language provides statements that can alter the flow of a sequence of instructions. These statements are called control statements.
- These statements help to jump from one part of the program to another. The control transfer may be conditional or unconditional.

Control Structure

- A control structure refers to the way in which the programmer specifies the order of executing the statements.
- Three control structures
  - Sequence structure
    - Programs are executed sequentially by default.
  - Selection structures
    - `if`, `if…else`, `switch`
  - Repetition structures (iteration)
    - `while`, `do…while`, `for`

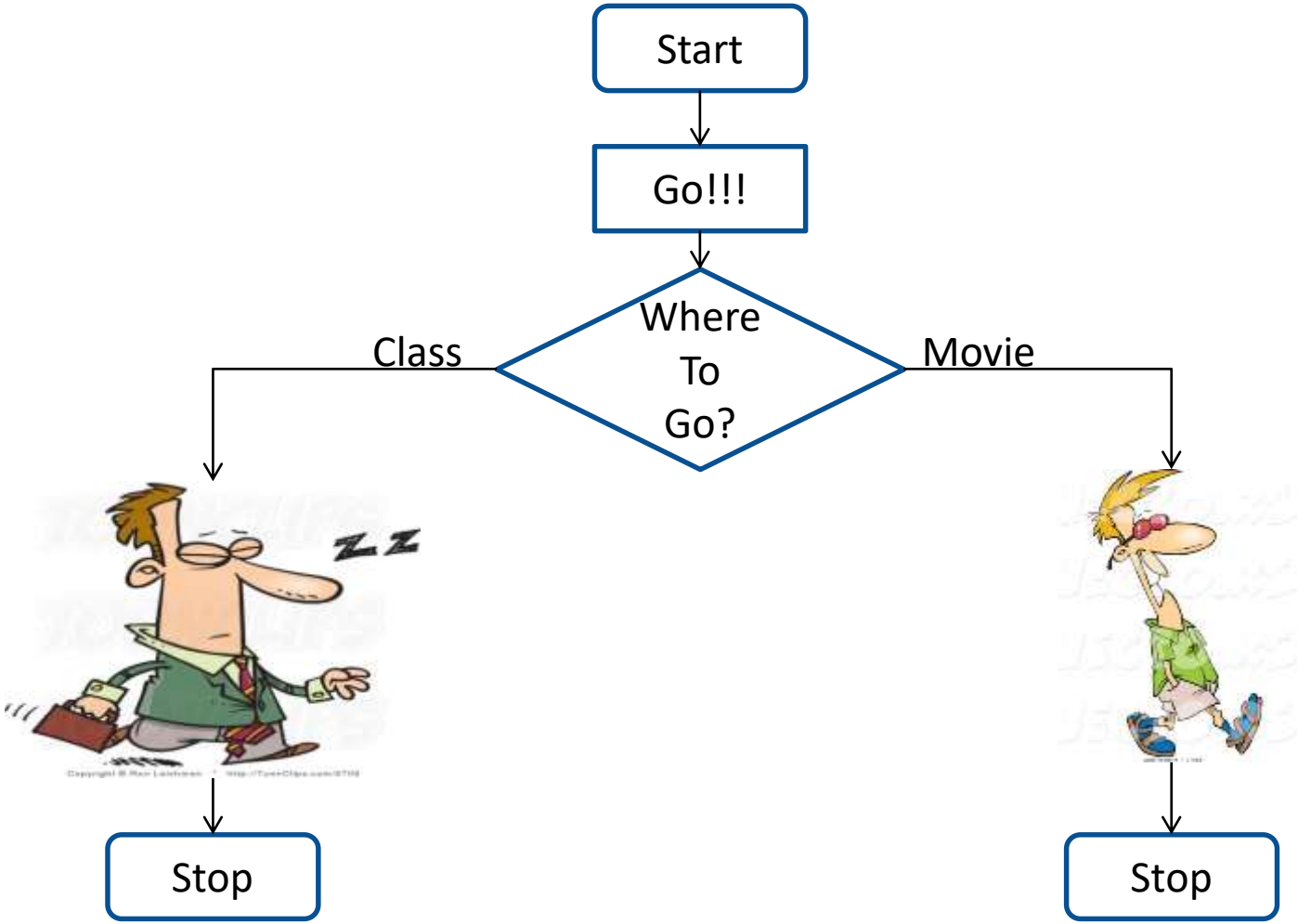




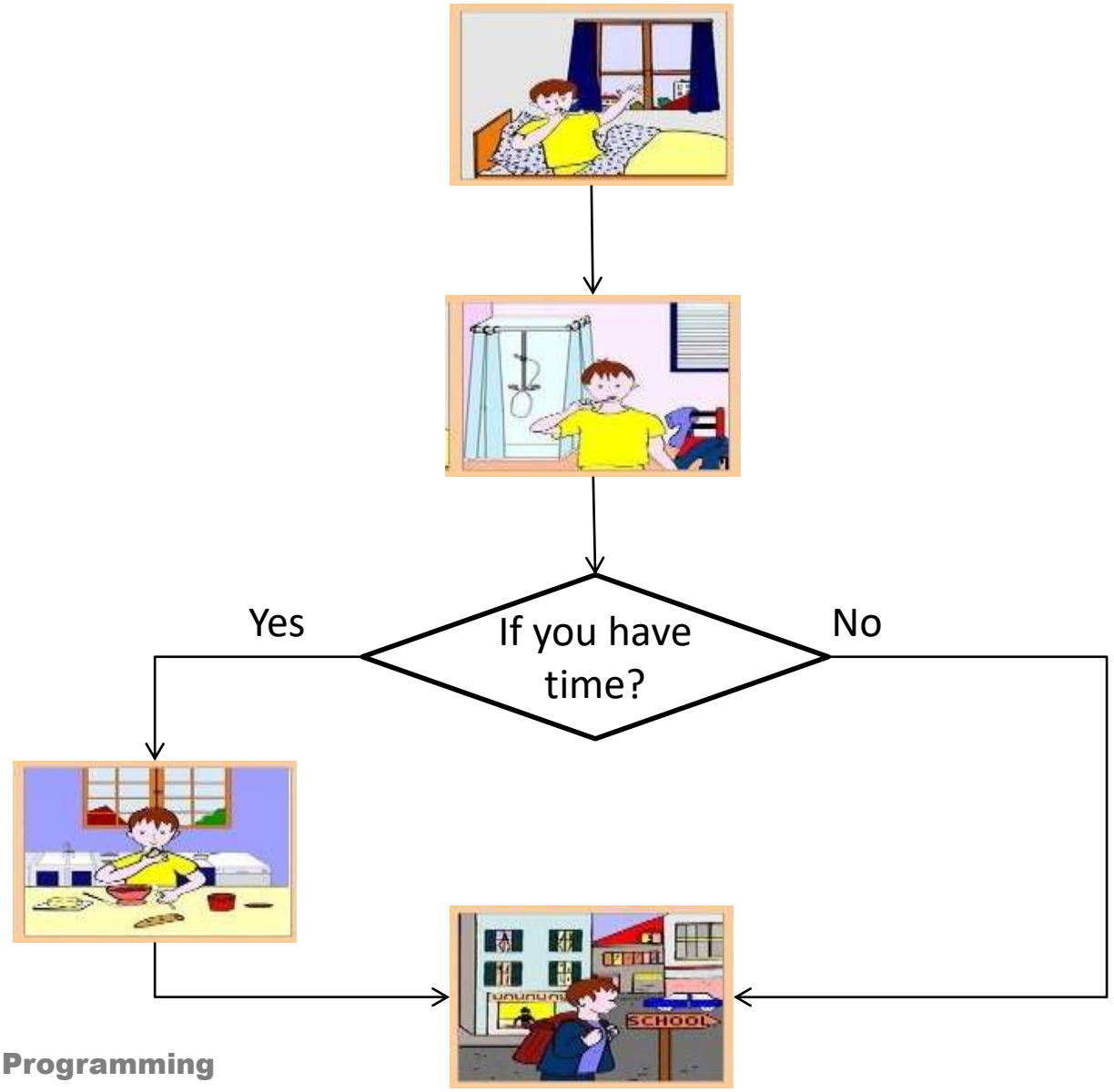
Condition Statements

- The C condition statements or the decision statements, checks the given condition
- Based upon the state of the condition, a sub-block is executed.
- Decision statements are the:
  - *if statement*
  - *if-else statement*
  - *switch statement*

Daily routine



`if` statement



`if` Statement

- If statement
  - It is decision making statement uses keyword `if`.
  - It allows the computer to evaluate the expression first
    - and then, depending on whether the value is 'true' or 'false', i.e. non zero or zero it transfers the control to a particular statement.

A decision can be made on any expression.
zero - `false`
nonzero - `true`
Example:
`3 < 4` is `true`

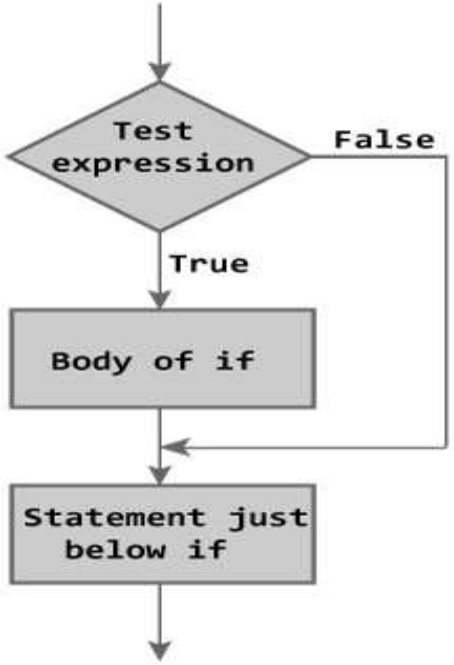
if Statement

Syntax

```
if (expression)
statement;

or

if (expression)
{
  block of statements;
}
```



`if` Statement

- The *if statement* has the following syntax:

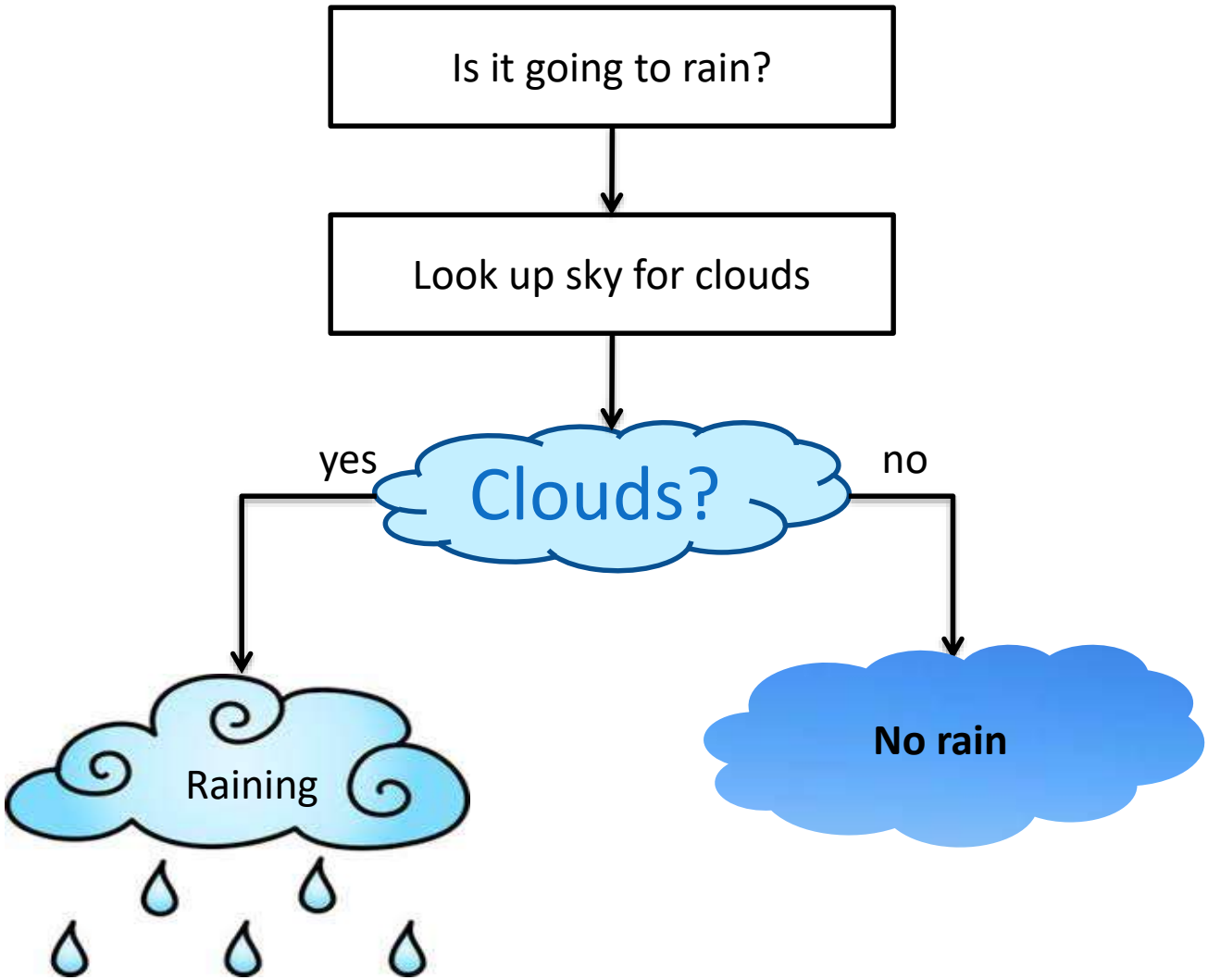
`if` is a C reserved word

The *`condition`* must be a boolean expression. It must Evaluate to either non-zero or zero.

```
if ( condition )/* no semi-colon */
    statement;
```

If the *`condition`* is non-zero, the *`statement`* is executed.
If it is zero, the *`statement`* is skipped.

Rain ???


Is it going to rain?
Look up sky for clouds
Clouds?
yes
no
Raining
No rain

```
#include<stdio.h>
void main()
{
 int v;
 printf("Enter the number :");
 scanf("%d", &v);
 if(v<10)
   printf("number is less  than 10");
}
```

Program to check whether number is less than 10.

```
Enter the number: 6
Number is less than 10
```

What will be the output of the following C code?

```
#include <stdio.h>
void main()
{
int x = 5;
if (x < 1)
printf("hello");
if (x == 5)
printf("hi");
else
printf("no");
}
```

a) hi b)hello c) no d) error

What will be the output of the following C code?

```
#include <stdio.h>
int x;
void main()
{
if (x)
printf("hi");
else
printf("how are u");
}
```

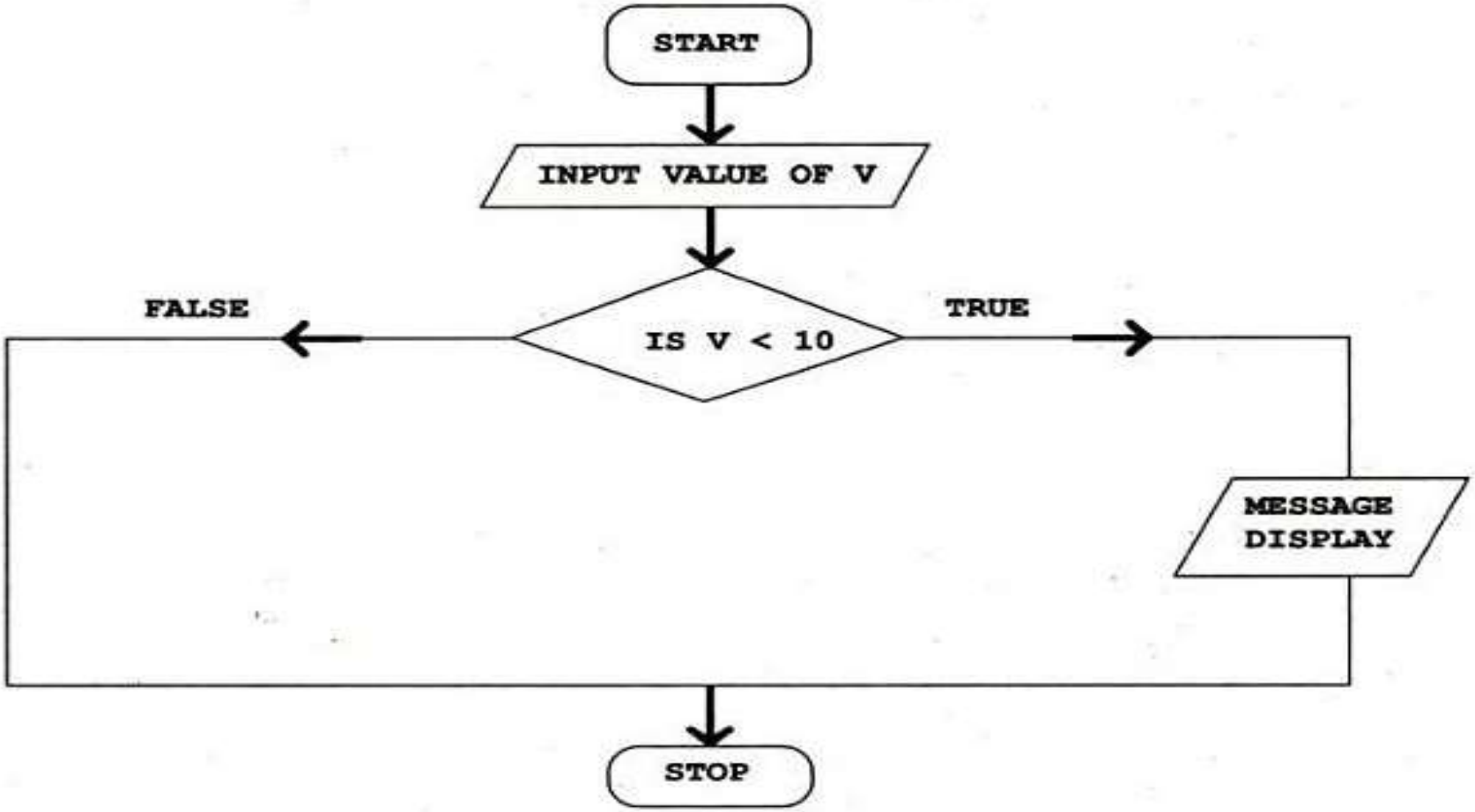
a) hi

b) how are you

c) compile time error

d) error

Control Flow



`if..else` statement

If you have time?

Yes

No

Grab something to eat along

`if..else` statement

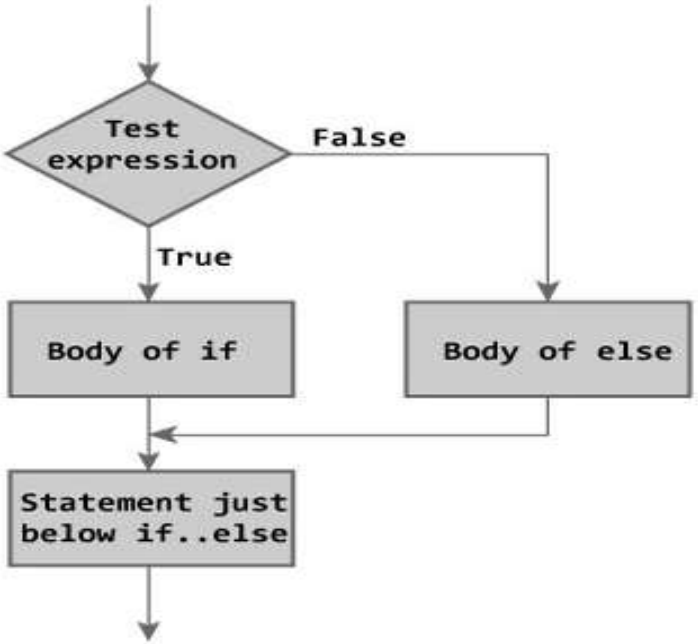
- The `if` statement executes only when the condition following if is true.
- It does nothing when the condition is false.
- The `if..else` statement takes care of the true and false conditions.

`if..else` statement

- `if..else` has two blocks.
- One block is for `if` and it is executed when condition is **non-zero**(true).
- The other block is of `else` and its executed when condition is **zero** (false).

Syntax

```
if (expression)
{
  block of statements;
}
else
{
  block of statements;
}
```



`if..else` statement

- The `else` statement cannot be used without `if`.
- No multiple `else` statements are allowed with one `if`.
- `else` statement has no expression.
- Number of `else` cannot be greater than number of `if`.

Ternary conditional operator (? :)

- C code:

```
if ( marks>= 60 )
   printf( "Pass\n");
else
   printf( "Fail\n");
```

- Same code using **ternary operator**:
  - Takes three arguments (condition, value if `true`, value if `false`)
  - Our code could be written:

```
printf("%s\n", grade >= 60 ? "Pass" : "Fail");
```

  - Or it could have been written:

```
grade >= 60 ? printf(“Pass\n”) :
  printf(“Fail\n”);
```

```
#include<stdio.h>
void main()
{
 int a;
 printf("Enter the number :");
 scanf("%d", &v);
 if(v<10)
   printf("number is less than 10");
 else
   printf("number is greater than 10");
}
```

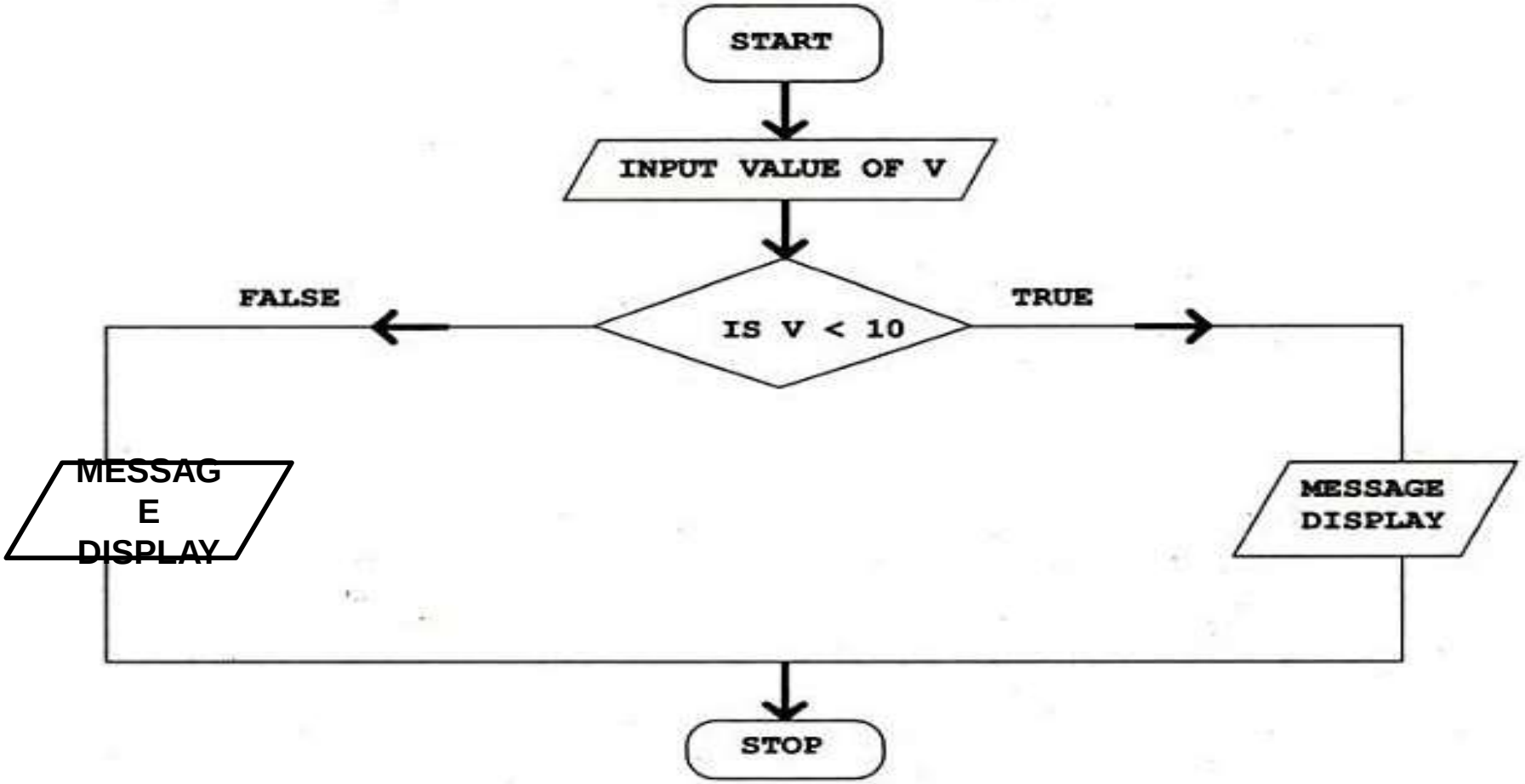
Example : Program to check whether number is less than 10.

```
Enter the number: 7
Number is less than 10
```

or

```
Enter the number: 100
Number is greater than 10
```

Control Flow



What will be the output of the following C code?

```
#include <stdio.h>
void main()
{
int x = 0;
if (x == 0)
printf("hi");
else
printf("how are u");
printf("hello");
}
```

a) hi b) how are you c) hello d) hihello

Nested `if..else`

- In `if..else` statement else block is executed by default after failure of if condition.
- The nested `if...else` statement is used when program requires more than one test expression.
- Test for multiple cases by placing `if…else` selection statements inside `if…else` selection statement.
- This kind of nesting will be unlimited.

Nested `if..else`

Syntax

```
if ( condition ) {
    block of statements;
}
else if ( condition ) {
    block of statements;
}
else {
    block of statements;
}
```

Program to check whether number is less than 10.

```
#include<stdio.h>
void main()
{
 int a;
 printf("Enter the number :");
 scanf("%d", &v);
 if(v<10){
   printf("number is less than 10");
 }
 else if(v<100){
   printf("number is less than 100");
 }
}
```

```
Enter the number: 1
Number is less than 10
```

or

```
Enter the number: 56
Number is less than 100
```

MCQ

If you have to make decision based on multiple choices, which of the following is best suited?

A. if

B. if-else

C. if-else-if

D. All of the above

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
int x = 0;
if (x == 1)
{
if (x == 0)
printf("inside if\n");
else
printf("inside else if\n");
}
else
printf("inside else\n");
}
```

a) inside if **b)** inside else if **c)** inside else **d)** compile time error

What will be output when you will execute following c code?

```
#include<stdio.h>
void main(){
   int a=100;
   if(a>10)
       printf("M.S. Dhoni");
   else if(a>20)
       printf("M.E.K Hussey");
   else if(a>30)
        printf("A.B. de villiers");
}
```

(A) M.S. Dhoni

(B) A.B. de villiers

(C) M.S Dhoni
M.E.K Hussey
A.B. de Villiers

D)none of the above

Forms of `if`

The **if** statement can take any of the following forms:

```
if ( condition )
        do this ;
or
if ( condition ) {
        do this ;
        and this ;
        }
```

```
if ( condition ) {
        do this ;
        and this ;
        }
else {
        do this ;
        and this ;
        }
```

```
if ( condition )
        do this ;
else
        do this ;
```

```
if ( condition )
   do this ;
else if ( condition )
   do this ;
else {
    do this ;
    and this ;
   }
```

Program to print grades of students marks.

```
#include<stdio.h>
void main()
{
 float marks;
 scanf("%f", &marks);
 if (marks>90){
      printf("Grade A");
    }
 else  if (marks>80) {
       printf("Grade B");
      }
 else  if(marks>70){
       printf("Grade C");
      }
 else if (marks >60) {
       printf("Grade D");
      }
}
```

```
66.70
Grade D
```

or

```
78.00
Grade C
```

Forms of `if`

Decision control statements	Syntax	Description
`if`	`if (condition){` `  Statements;}`	In these type of statements, if condition is true, then respective block of code is executed.

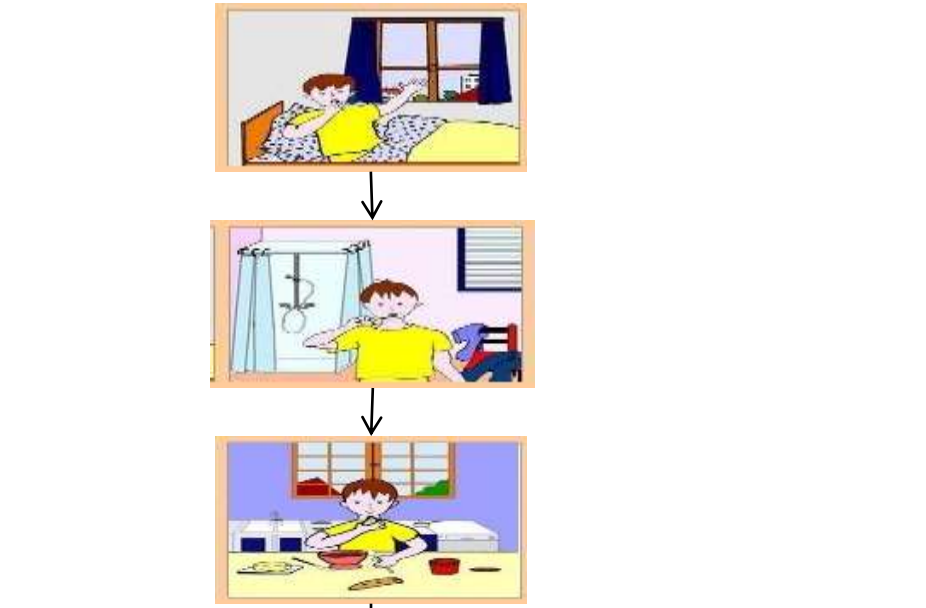
`if…else`	`if (condition){`
	`  Statement1;`
	`  Statement2;}`
	`else {`
	`  Statement3;`
	`  Statement4;}`
		In these type of statements, group of statements are executed when condition is true. If condition is false, then else part statements are executed.

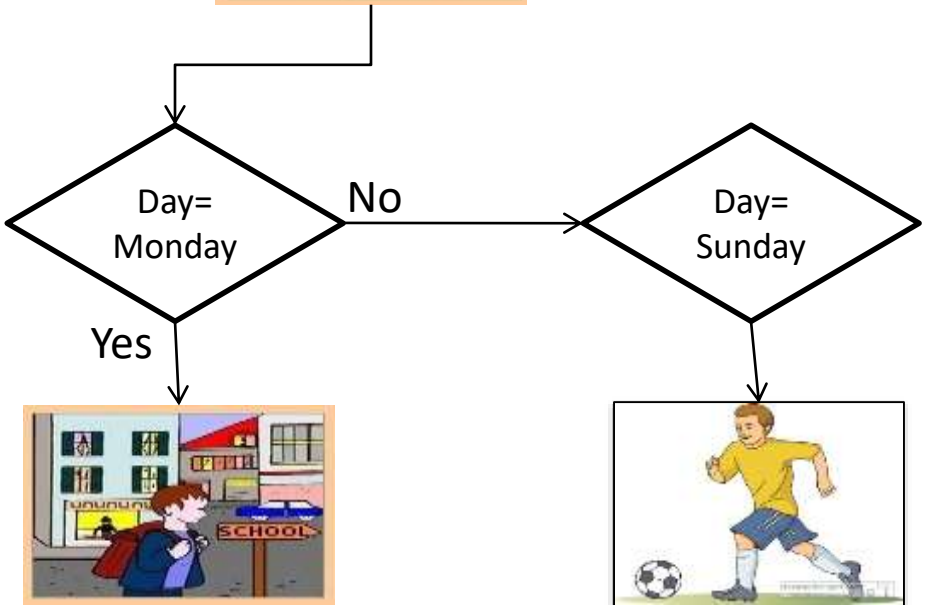
Nested `if`	`if (condition1){`
	`  Statement1;}`
	`else`
	`if(condition2){`
	`  Statement2;}`
	`else`
	`  Statement 3;`
		If condition 1 is false, then condition 2 is checked and statements are executed if it is true. If condition 2 also gets failure, then else part is executed.

`break` statement

- **`break`** is a **keyword**.
- `break` allows the programmer to terminate the loop.
- A `break` statement causes control to transfer to the first statement after the loop or block.
- The `break` statement can be used in nested loops. If we use `break` in the innermost loop then the control of the program is terminated only from the innermost loop.

`switch` Statement





`switch` Statement

- The control statement that allows to make a decision from the number of choices is called switch.
- Also called switch-case-default.
- The switch statement provides another way to decide which statement to execute next.
- The switch statement evaluates an expression, then attempts to match the result to one of several possible cases.
- Each case contains a value and a list of statements.
- The flow of control transfers to statement associated with the first case value that matches.

switch Statement

Syntax

```
switch (expression)
{
case constant1:
        statements;
        break;
case constant2:
        statements;
        break;
case constant3:
        statements;
        break;
default:
        statements;
}
```

switch and case are reserved words

```
switch ( expression )
{
   case value1 :
      statement-list1
   case value2 :
      statement-list2
   case value3 :
      statement-list3
   case  ...

}
```

If *expression* matches *value2*, control jumps to here

Rules of using `switch case`

1. Case label must be **unique**
2. Case label must end with colon
3. Case label must have **constant expression**
4. Case label must be of integer, character type like case 2, case 1+1, case ‘a’
5. Case label should not be **floating point**
6. Default can be placed anywhere in switch
7. Multiple cases cannot use same expression
8. **Relational operators** are not allowed in switch
9. Nesting of switch is allowed.
10. **Variables** are not allowed in switch case label..

Syntax error in `switch` statement

```
switch(pt){
      case count:
       printf("%d", count);
       break;
      case 1<8:
       printf("A point");
       break;
      case 2.5:
       printf("A line");
       break;
      case 3 + 7.7:
       printf("A triangle");
      case 3 + 7.7:
       printf("A triangle");
       break;
      case count+5:
       printf("A pentagon");
       break;
}
```

Variable cannot be used as label

Relational operators are not allowed

Floating point number cannot be used

Floating point number cannot be used and same expression cannot be used

constant expression should be used

What is the output of this C code(When 1 is entered)?

```
void main()
{
   int ch;
   printf("enter a value btw 1 to 2:");
   scanf("%d", &ch);
   switch (ch)
   {
   case 1:
      printf("1\n");
   default:
      printf("2\n");
   }
}
```

A. 1 **B.** 2 **C.** 1 2 **D.** Run time error

What is the output of this C code(When 1 is entered)?

```
void main()
{
   int ch;
   printf("enter a value btw 1 to 2:");
   scanf("%d", &ch);
  switch (ch, ch + 1)
   {
   case 1:
      printf("1\n");
      break;
   case 2:
      printf("2");
     break;
   }
}
```

A. 1 **B. 2** **C. 3** **D. Run time error**

Program to show switch statement in geometry

```
#include<stdio.h>
void main()
{
   int pt;
   printf("Enter the number of nodes:");
   scanf("%d", &pt);
   switch(pt){
     case 0:
      printf("\nNo Geometry");
     break;
     case 1:
      printf("\nA point");
     break;
     case 2:
      printf("\nA line");
     break;
     case 3:
      printf("\nA triangle");
     break;
     case 4:
      printf("\nA rectangle");
     break;
     case 5:
      printf("\nA pentagon");
     break;
    default:
     printf("Invalid input");
    }
}
```

```
Enter the number of nodes: 2
A line
```

Program to move a car in car game

```
#include<stdio.h>
void main()
{
   int key;
   printf("Press 1 to turn left.");
   printf("Press 2 to turn right.");
   printf("Press 3 to increase speed.");
   printf("Press 4 for break: ");
   scanf("%d", &key);
   switch(key){
     case 1:
       printf("\nTurn left");
     break;
     case 2:
       printf("\nTurn right");
     break;
     case 3:
       printf("\nIncrease speed");
     break;
     case 4:
      printf("\nBreak");
     break;
     default:
       printf("Invalid input");
    }
```

```
Press 1 to turn left.
Press 2 to turn right.
Press 3 to increase speed.
Press 4 for break: 4
Break
```

Next Class: Loop control and Jump statements

cse101@lpu.co.in

CSE101-Lec 7

Control Structures

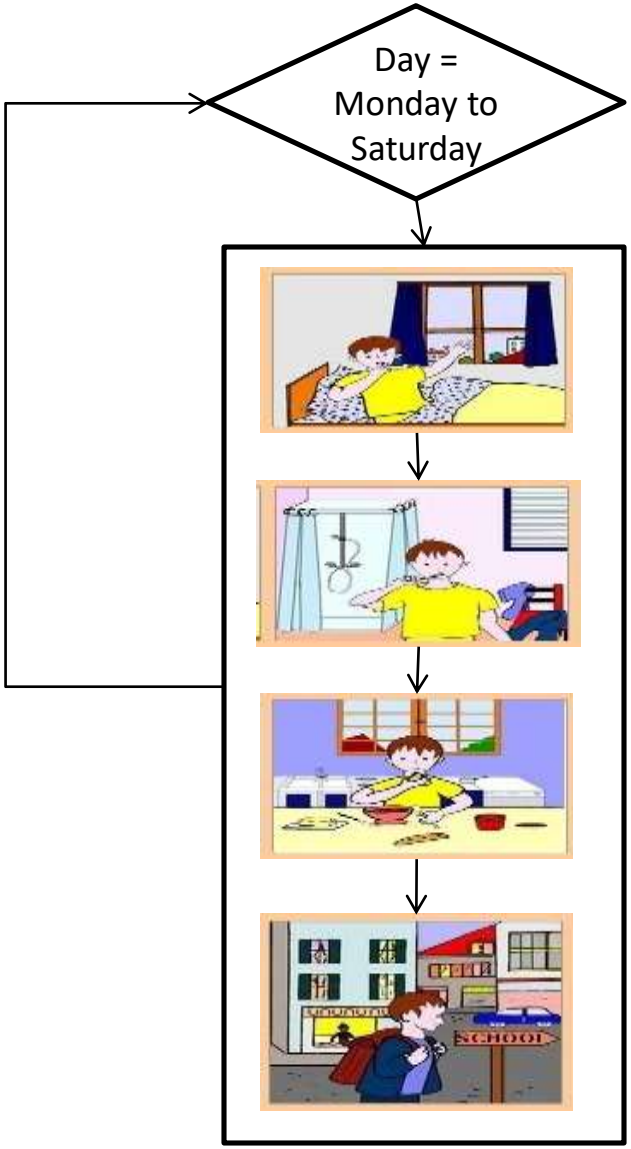
(Repetition structure)

Jump Statements

Outline

- Repetition structure/Control Loop Statements
  - for statement
  - while statement
  - do-while statement
- Jump Statements
  - break
  - continue
  - goto
  - return

Repetition(Going to School)



Repetition Statement

- A **repetition statement** allows you to specify that an action is to be repeated while some condition remains true.

Looping (repetition)

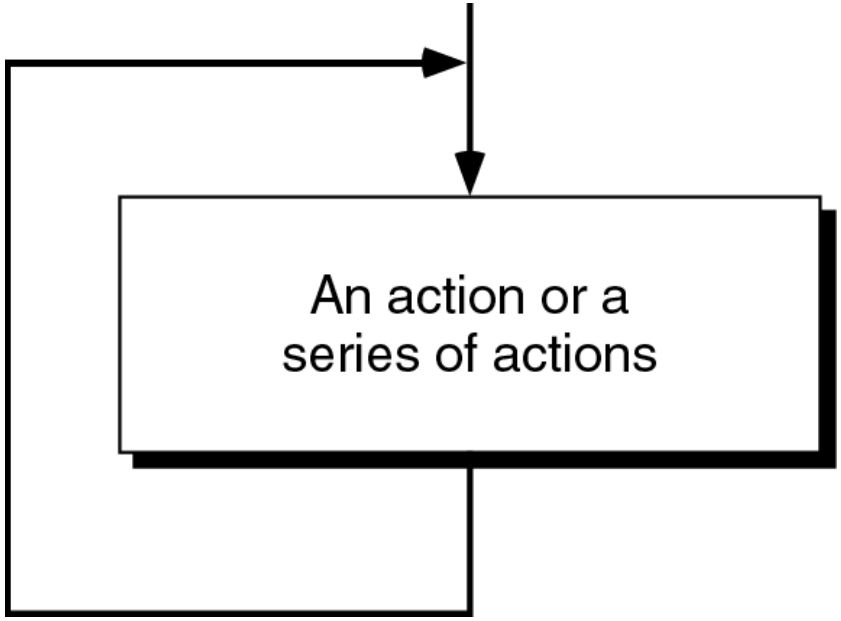
- *What if we want to display hello 500 times?*
  - Should we write 500 printf statements or equivalent ?

➢ Obviously not.

- It means that we need some programming facility to repeat certain works.
- Such facility is available in form of ***looping statements.***

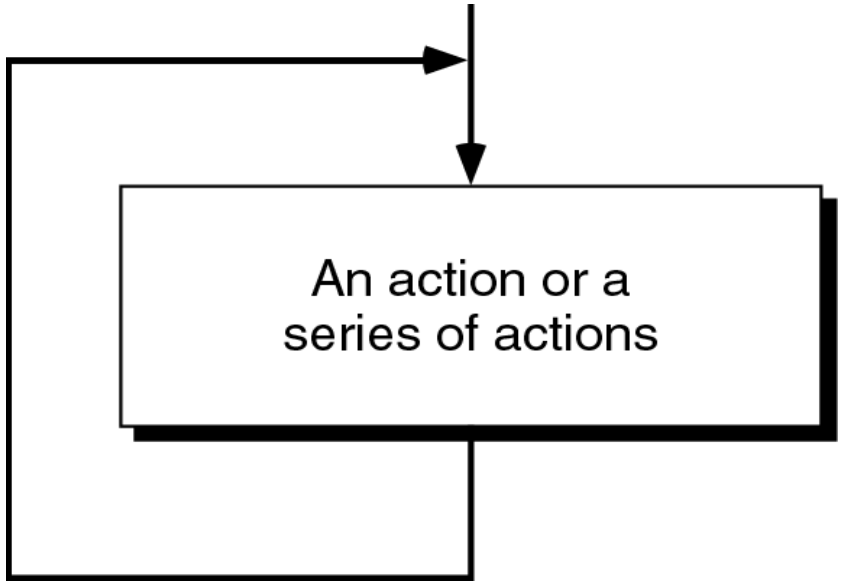
Loop

- The main idea of a loop is to repeat an action or a series of actions.



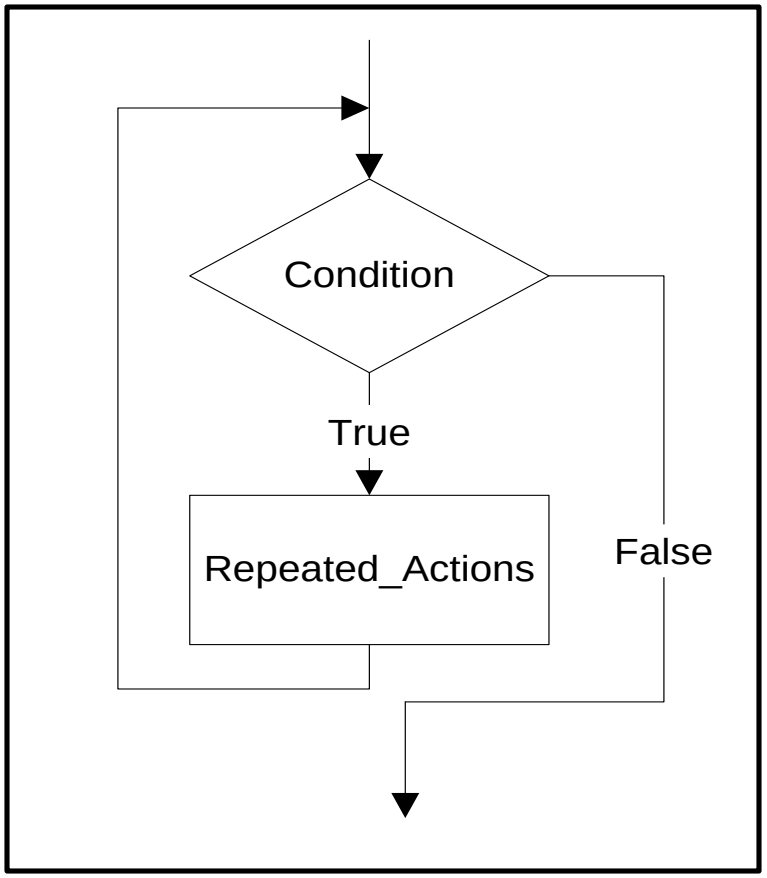
The concept of a loop without condition

- But, when to stop looping?
- In the following flowchart, the action is executed over and over again. It never stops – This is called an infinite loop
- **Solution** – put a condition to tell the loop either continue looping or stop.



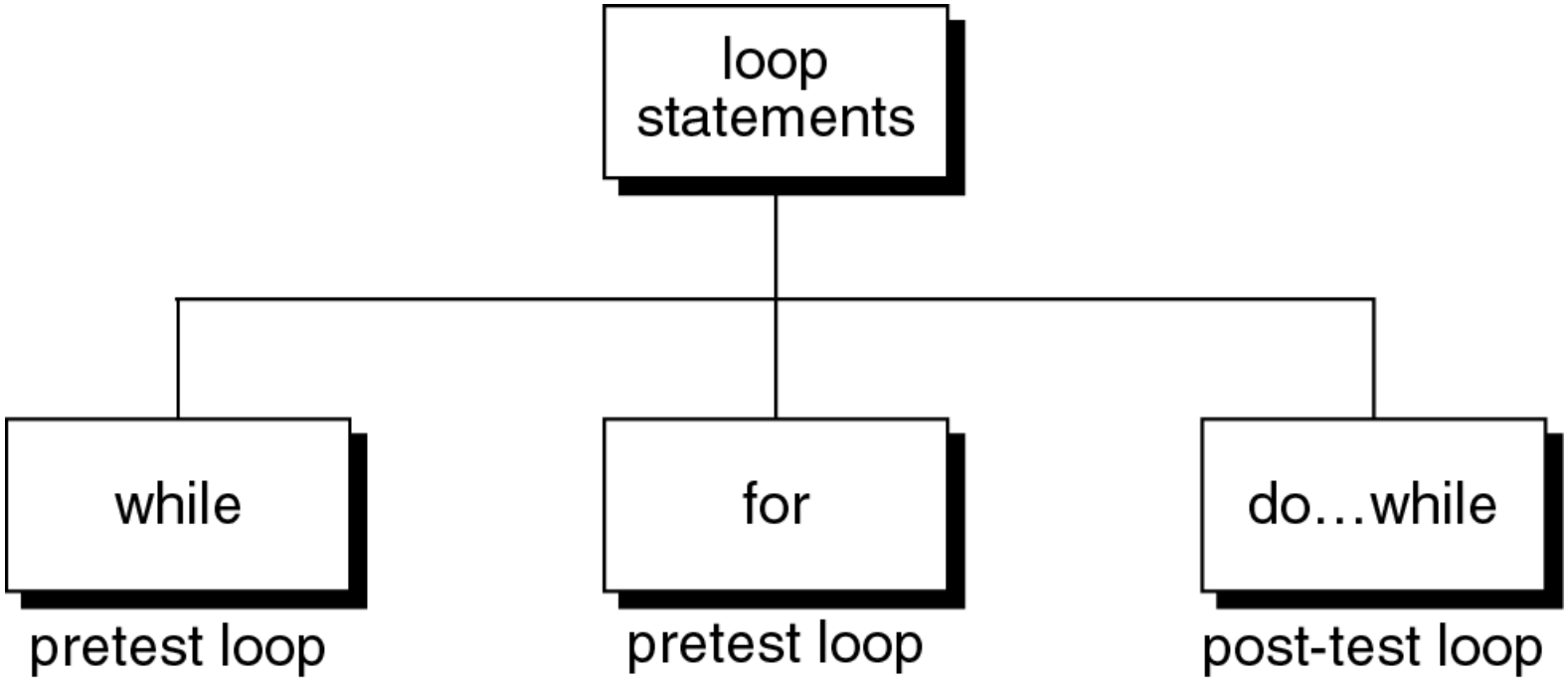
Loop

- A loop has two parts – **body** and **condition**
- **Body** – a statement or a block of statements that will be repeated.
- **Condition** – is used to control the iteration – either to continue or stop iterating.



Loop statements

- C provides three loop statements:

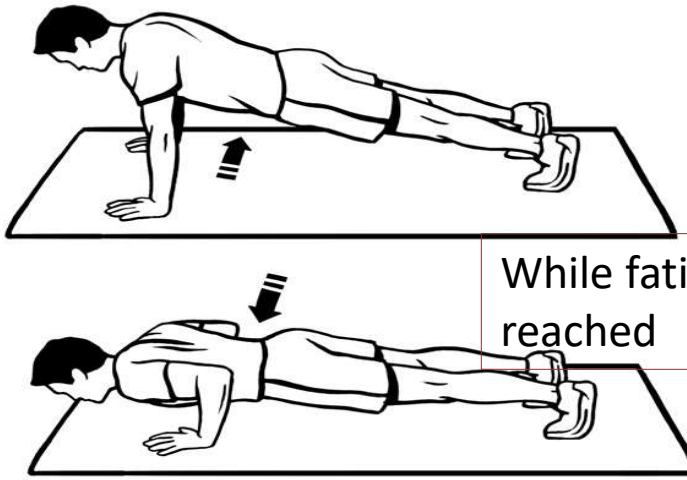


The "while" Statement in C

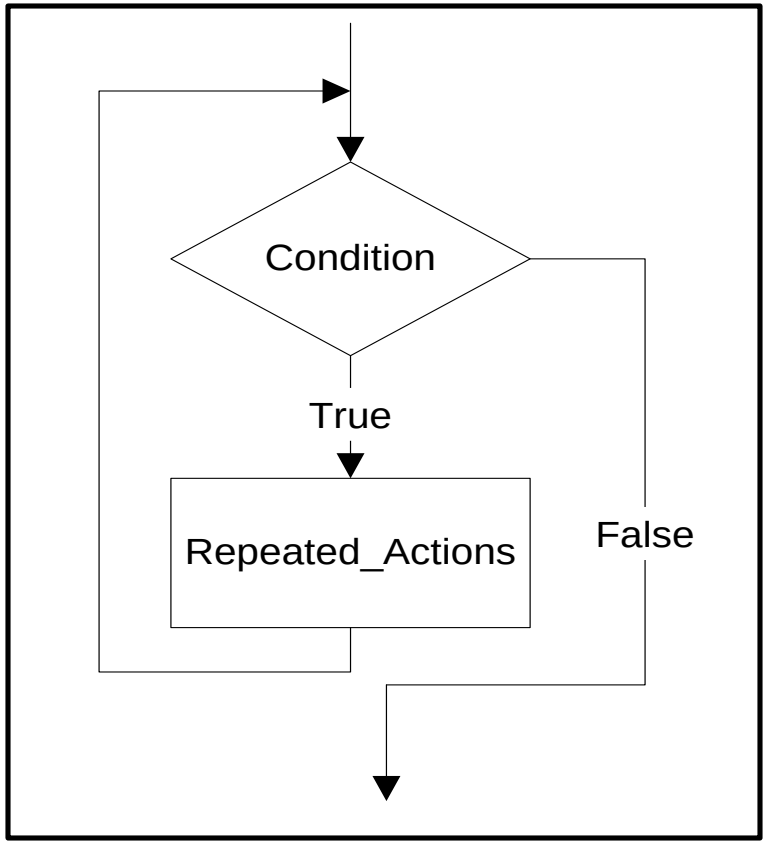
- The syntax of **while** statement in C:

Syntax

```
while (loop repetition condition){
   statement;
   updating control;
}
```



While fatigue level is not reached



while statement

```
while(loop repetition condition)
{
   Statements;
}
```

Loop repetition condition is the condition which controls the loop.

- The ***statement*** is repeated as long as the loop repetition condition is **true**.
- A loop is called an **infinite loop** if the loop repetition condition is always true.

What will be the output of the C program?

```
#include<stdio.h>
 int main()
{
int i = 4,j = 7;
while(++i < --j)
printf("Loop")
return 0;
}
```

a) Loop b) Loop Loop c) Loop Loop Loop d)infinite loop

What will be the output of the C program?

```
#include<stdio.h>
 int main()
{
int i = 4;
while(i == 4--)
printf("Loop ");
return 0;
 }
```

A. Loop Loop Loop Loop

B. Loop Loop loop

C. Compilation Error

D. Prints Nothing

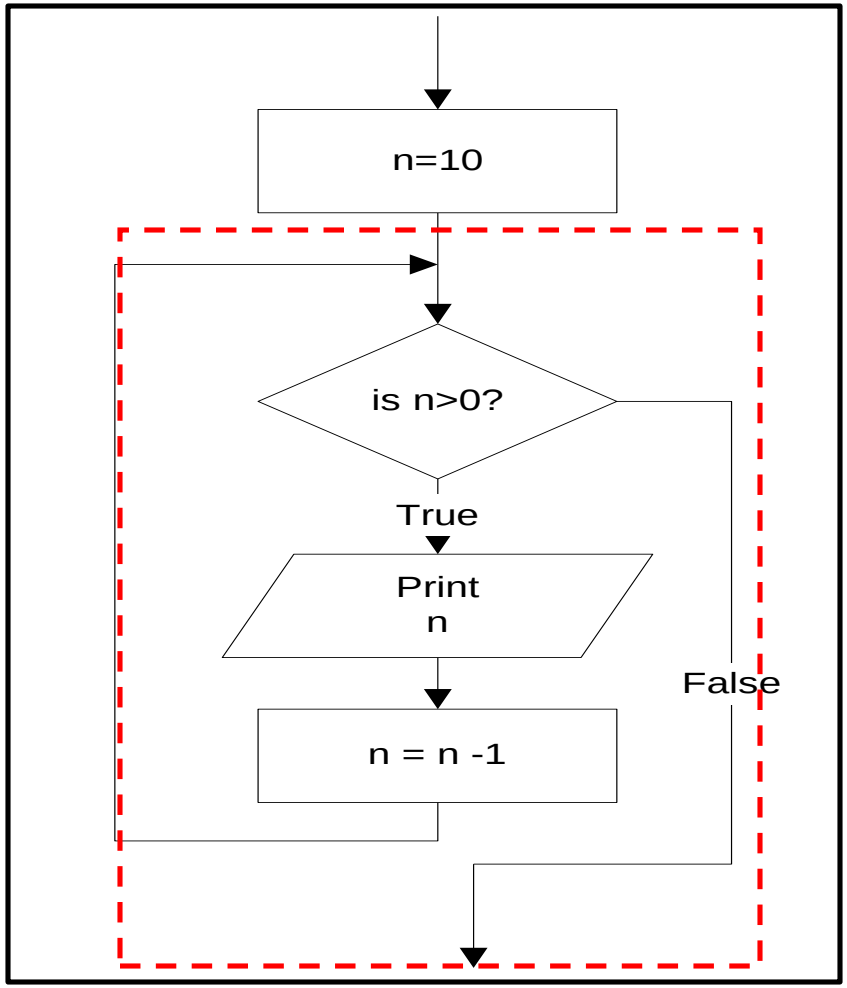
while statement

Example: This while statement prints numbers 10 down to 1

```
#include<stdio.h>
int main()
{
 int n=10;
 while (n>0){
  printf("%d ", n);
  n=n-1;
 }
}
```

```
10 9 8 7 6 5 4 3 2 1
```

count condition



The for Statement in C

- The syntax of `for` statement in C:

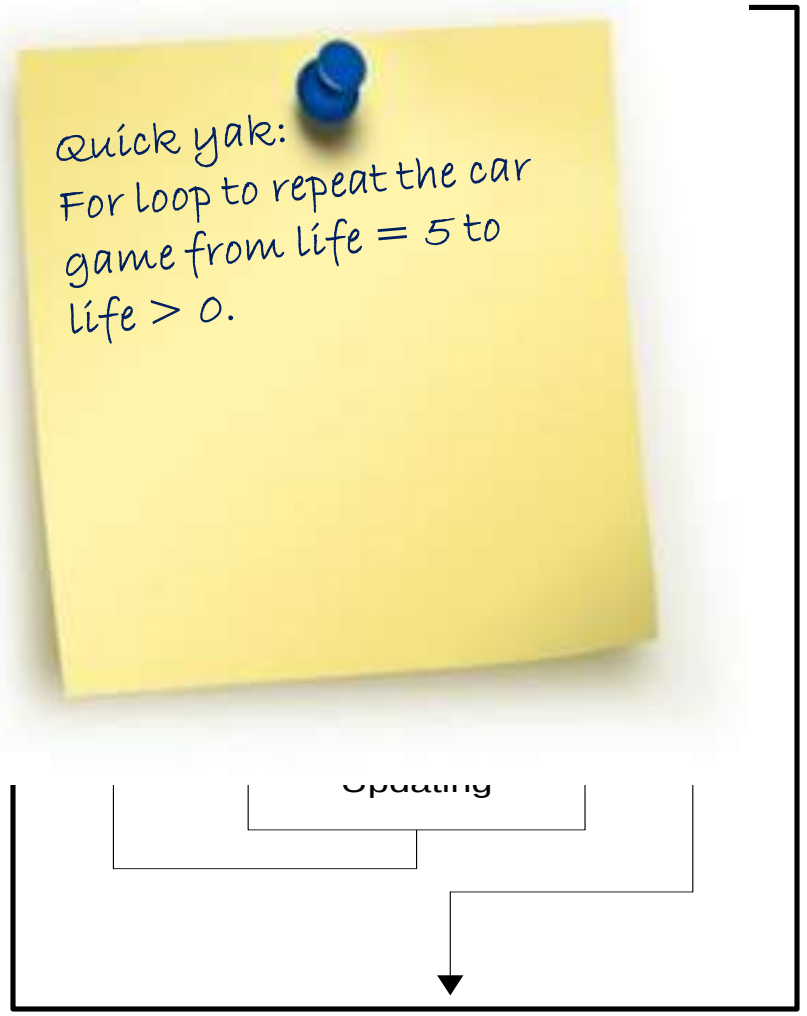
Syntax

```
for (initialization-expression;
        loop-repetition-condition;
        update-expression){
  statement;
}
```

- The **initialization-expression** set the initial value of the loop control variable.
- The **loop-repetition-condition** test the value of the loop control variable.
- The **update-expression** update the loop control variable.

for statement

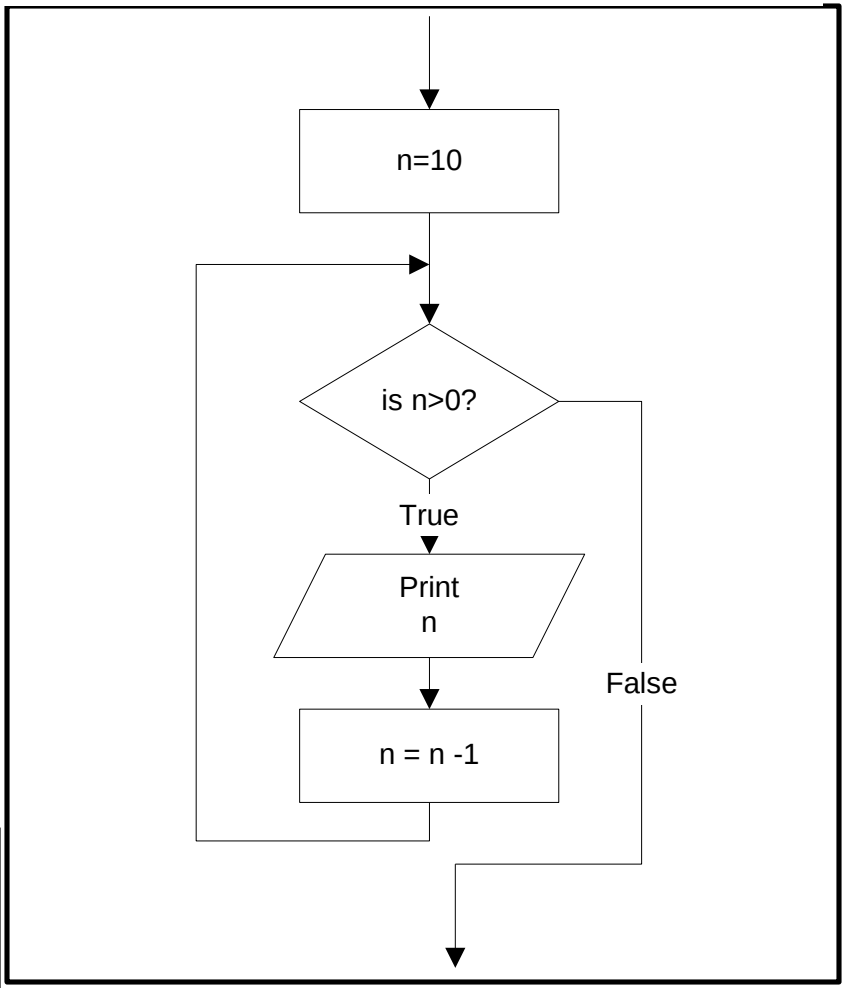
```
for (Initialization; Condition; Updating)
{
   Repeated_Actions;
}
```



for statement

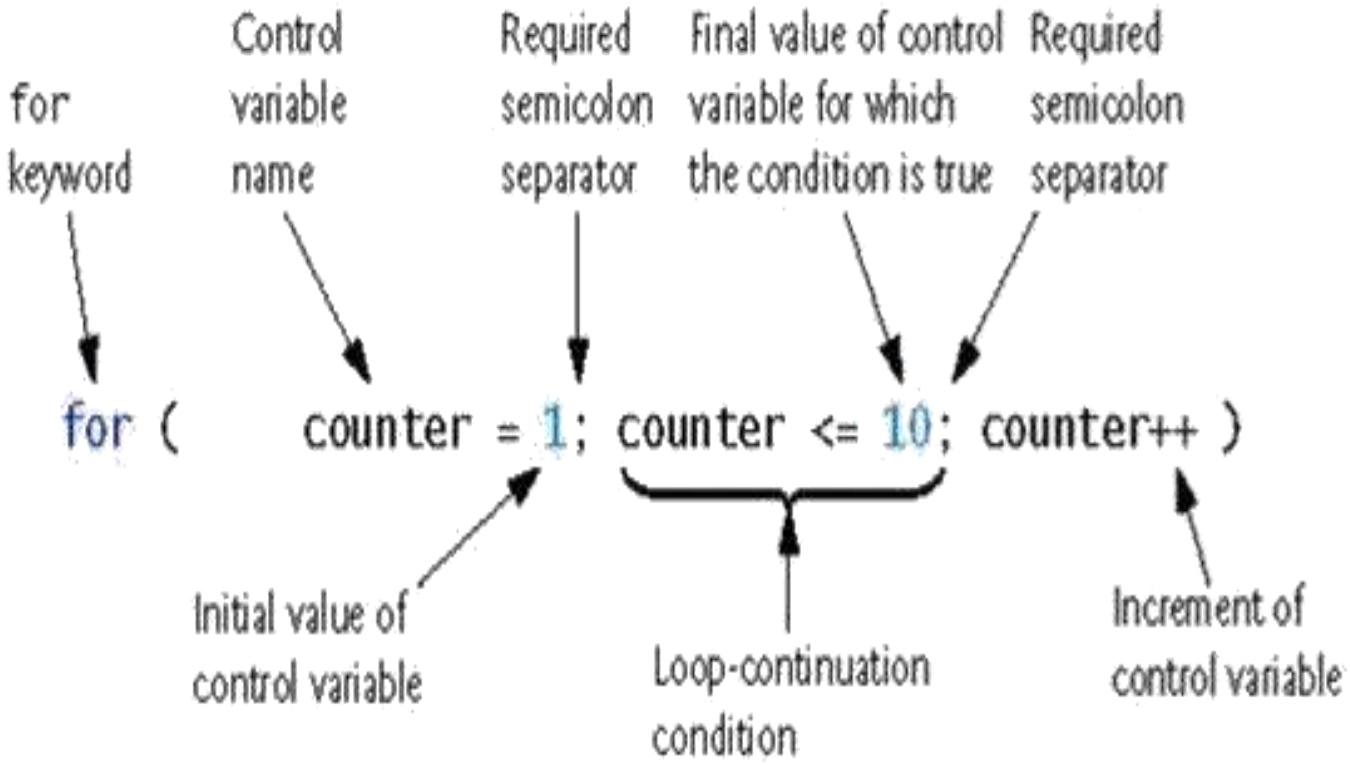
Example: This for statement prints numbers 10 down to 1

```
#include<stdio.h>
int main()
{
 int n;
 for (n=10; n>0; n=n-1){
  printf("%d ", n);
 }
}
```




```
10 9 8 7 6 5 4 3 2 1
```

Do TEN push ups = for count=1; count<=10; count++


for keyword
Control variable name
Required semicolon separator
Final value of control variable for which the condition is true
Required semicolon separator
for (counter = 1; counter <= 10; counter++)
Initial value of control variable
Loop-continuation condition
Increment of control variable

Nested Loops

- Nested loops consist of an **outer loop** with one or more **inner loops**.
  - Eg:

```
for (i=1;i<=100;i++){
      for(j=1;j<=50;j++){
            …
      }
}
```

Outer loop

Inner loop

- The above loop will run for 100*50 iterations.

Program to print tables up to a given number.

```
#include<stdio.h>
void main()
{
 int i,j,k ;
 printf("Enter a number:");
 scanf("%d", &k);
 printf("the tables from 1 to %d: \n",k);
 for(i=1; i<k; i++){
   for(j=1; j<=10; j++){
     printf("%d ",i*j);
    } //end inner for loop
   printf("\n");
 } //end outer for loop
getch();
} //end main
```

```
Enter a number
4
The tables from 1 to 4
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
```

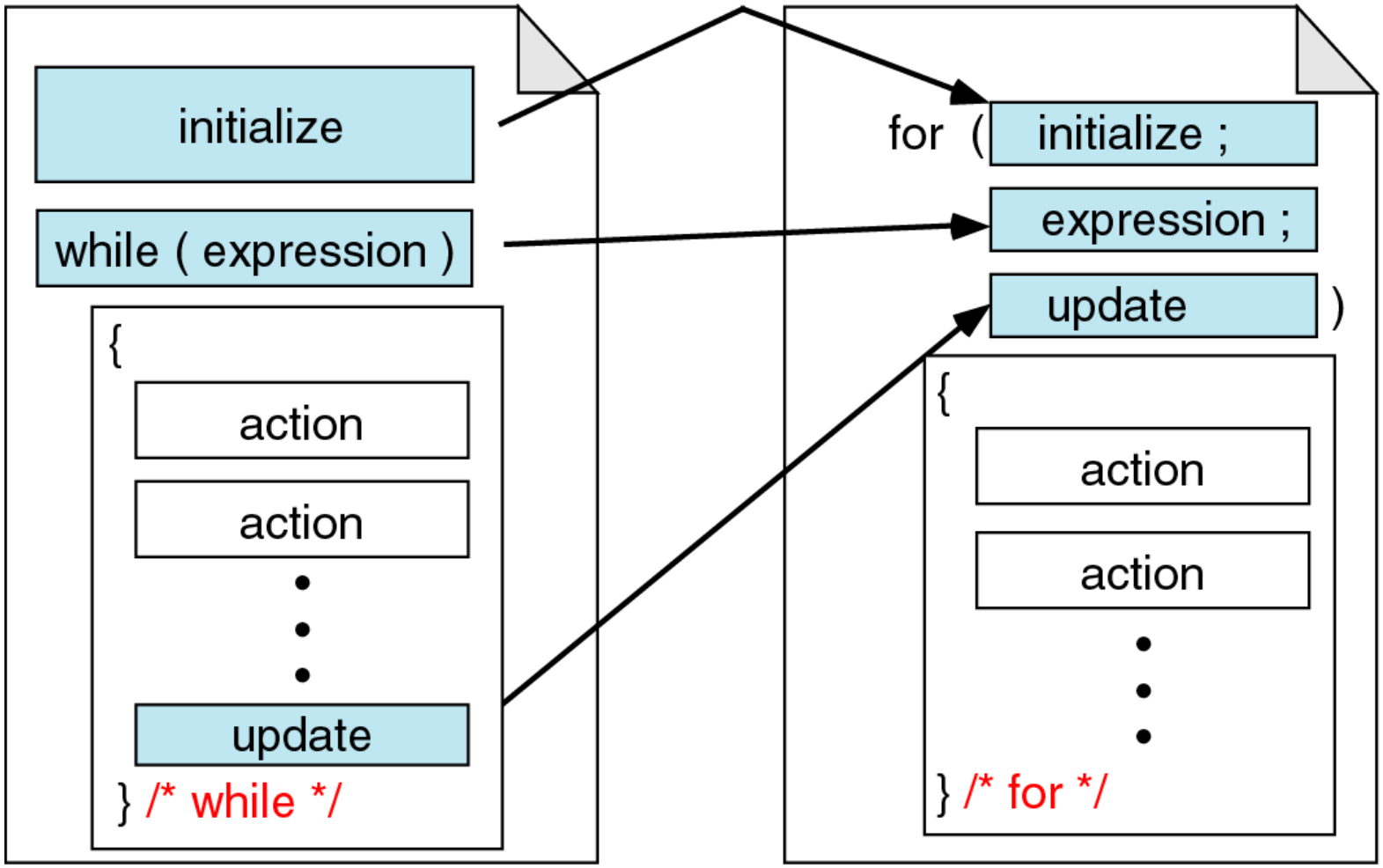
Program to display a pattern.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j;
 printf("Displaying right angled triangle for 5 rows");
 for(i=1 ; i<=5 ; i++) {
   for(j=1 ; j<=i ; j++)
       printf("* ");
   printf("\n");
 }
}
```

Displaying right angled triangle for 5 rows

```
*
* *
* * *
* * * *
* * * * *
```

While vs. for statements



Comparing `for` and `while` loops

The `do-while` Statement in C

- The syntax of do-while statement in C:

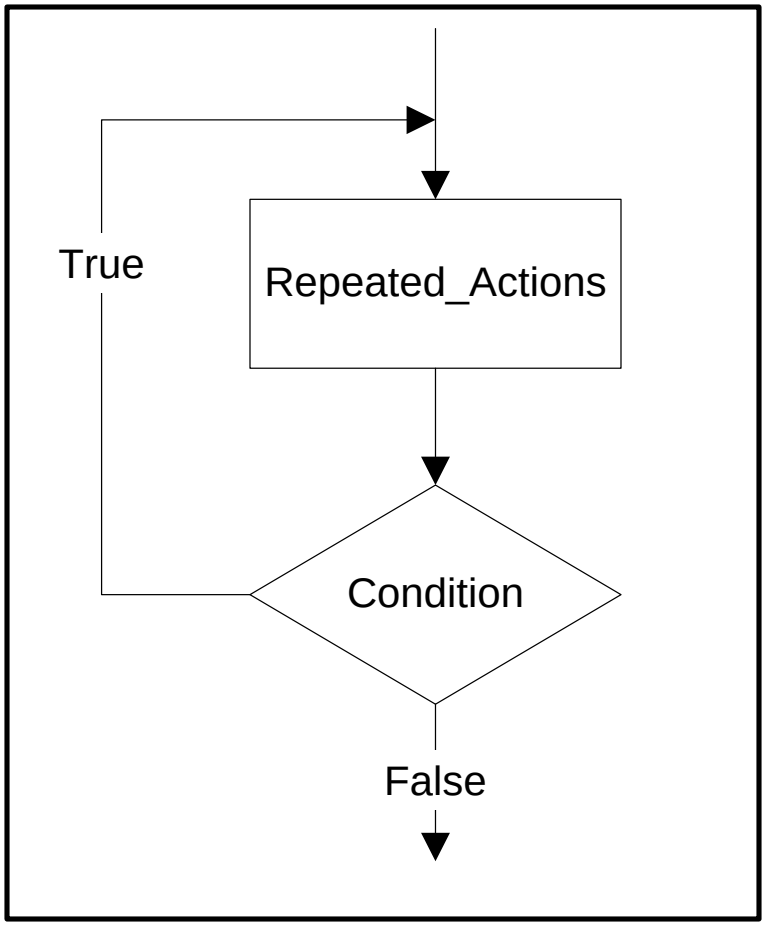
Syntax

```
do
  {
    statement;
  }while (condition);
```

- The *statement* executed at least one time.
- For second time, If the **condition** is true, then the *statement* is repeated else the loop is exited.

do…while statement

```
do
{
   Repeated_Actions;
} while (Condition);
```

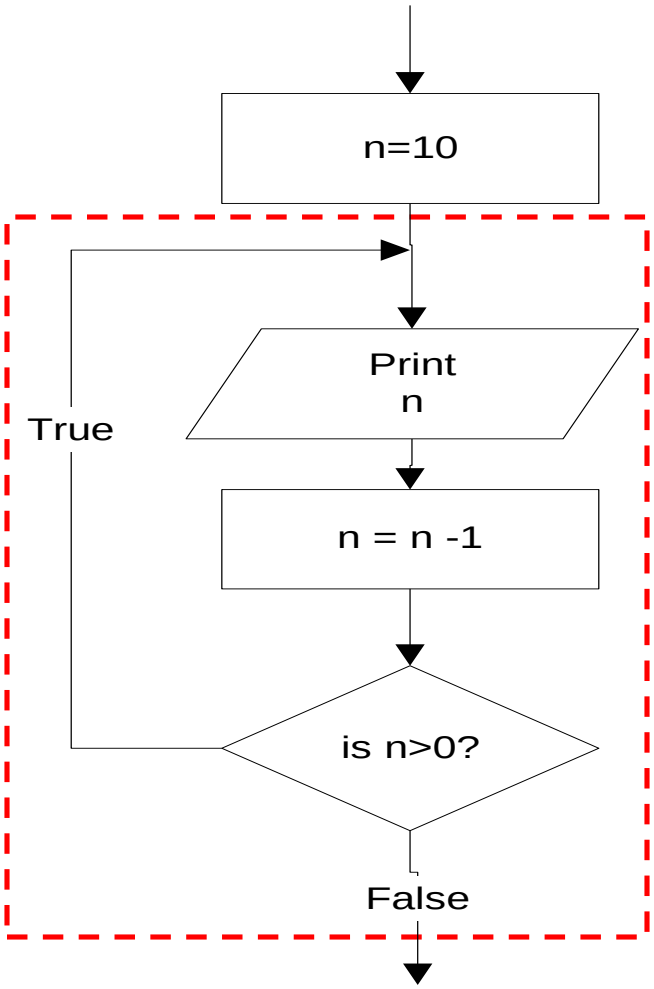


do…while statement

Example: this do…while statement prints numbers 10 down to 1

```
#include<stdio.h>
int main()
{
 int n=10;
 do{
  printf("%d ", n);
  n=n-1;
 }while (n>0);
}
```

```
10 9 8 7 6 5 4 3 2 1
```

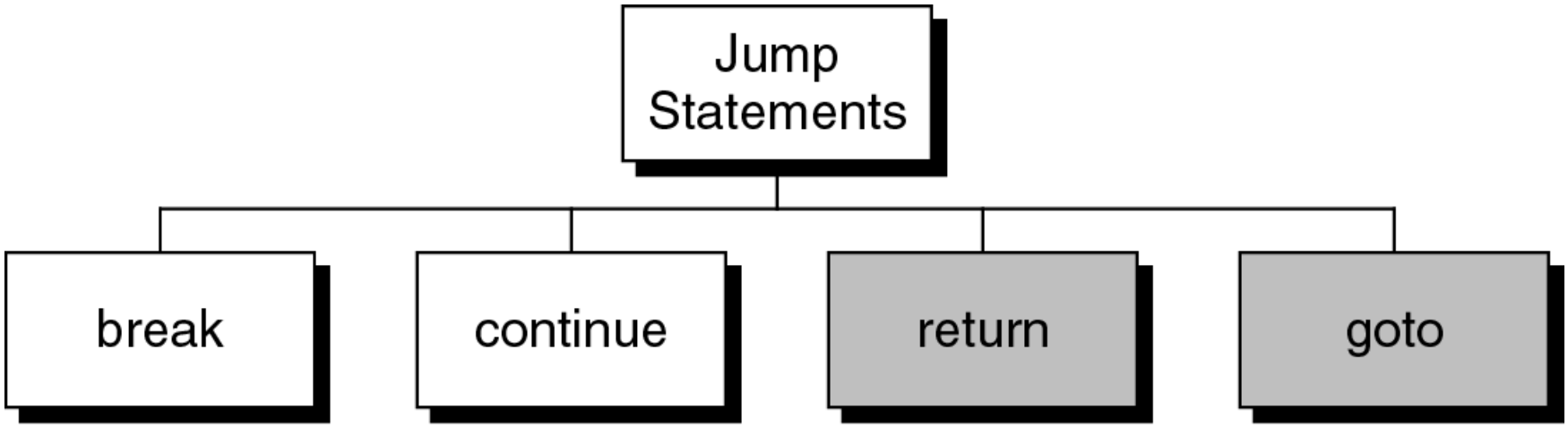


Difference between while and do..while

while loop	do..while loop
1. Condition is specified at the **top**	1. Condition is mentioned at the **bottom**
2. Body statements are executed when the condition is satisfied	2. Body statements are executed at least once even if the expression value evaluates to false
3. It is an **entry controlled** loop	3. It is an **exit controlled** loop
4.Syntax: **while** (**condition**) *statement;*	4.Syntax: **do** **{** *statements;* **}** **while** (**condition);**

Jump statements

- You have learn that, the repetition of a loop is controlled by the loop condition.
- C provides another way to control the loop, by using **jump statements.**
- There are four jump statements:



`break` statement

- `break` is a keyword.
- `break` allows the programmer to **terminate** the loop.
- A `break` statement causes control to transfer to the first statement after the loop or block.
- The `break` statement can be used in nested loops. If we use `break` in the innermost loop then the control of the program is terminated only from the innermost loop.

break statement

```
##include<stdio.h>
int main()
{
 int n;
 for (n=10; n>0; n=n-1){
  if (n<8)
   break;
  printf("%d ", n);
 } //end for
}
```

Program to show use of break statement.

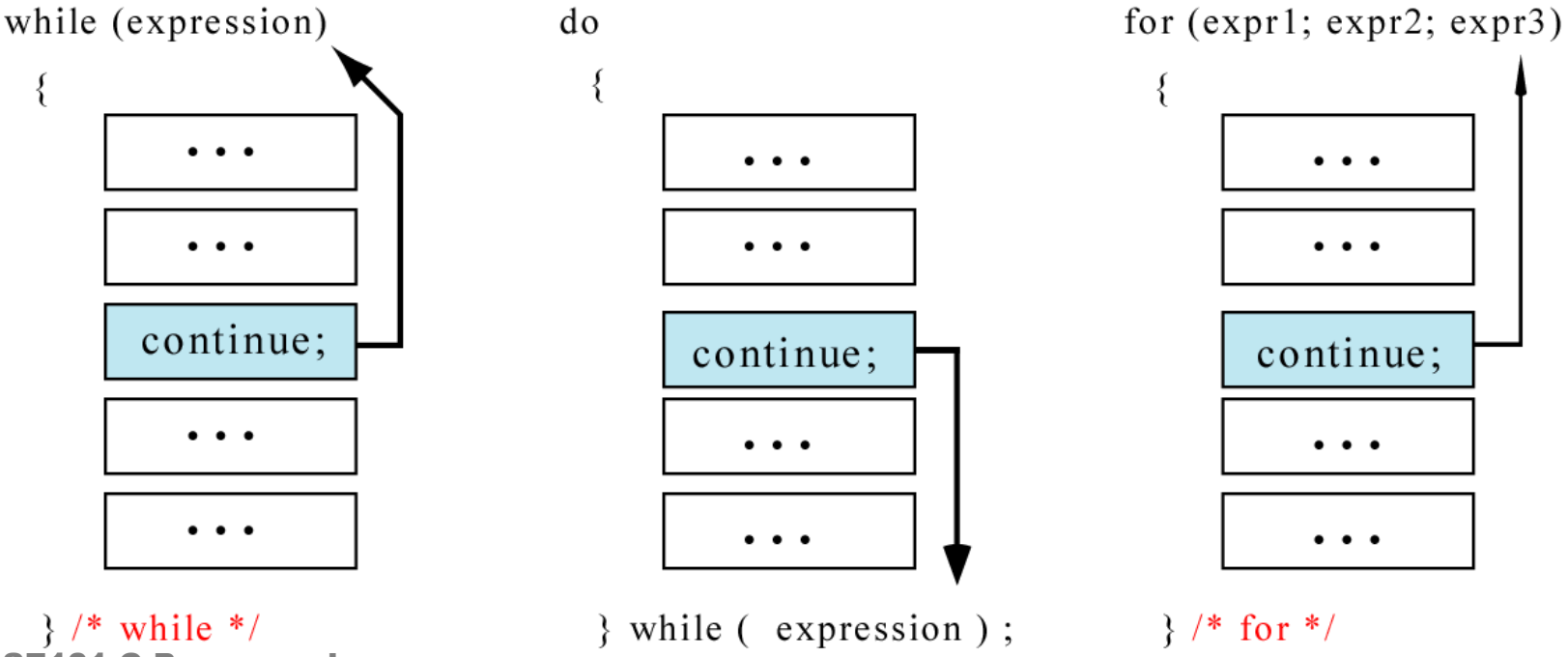
```
10 9 8
```

`continue` statement

- `continue` statement is exactly opposite to `break`.
- `continue` statement is used for continuing the next iteration of the loop statements
- When it occurs in the loop, it does not terminate, but skips the statements after this statement

`continue` statement

- In `while` and `do…while` loops, the continue statement transfers the control to the loop condition.
- In `for` loop, the continue statement transfers the control to the updating part.



continue statement

```
#include<stdio.h>
int main()
{
 int n;
 for (n=10; n>0; n=n-1){
  if (n%2==1)
     continue;
   printf("%d ", n);
 }
}
```

```
10 8 6 4 2
```

Program to show the use of continue statement in for loop

continue statement

```
#include<stdio.h>
int main()
{
 int n = 10;
 while(n>0){
  printf("%d", n);
  if (n%2==1)
    continue;
  n = n -1;
 }
}
```

For n=9, loop goes to infinite execution

Program to show the use of continue statement in for loop

```
10 9 9 9 9 9 ……
```

The loop then prints number 9 over and over again. It never stops.

goto

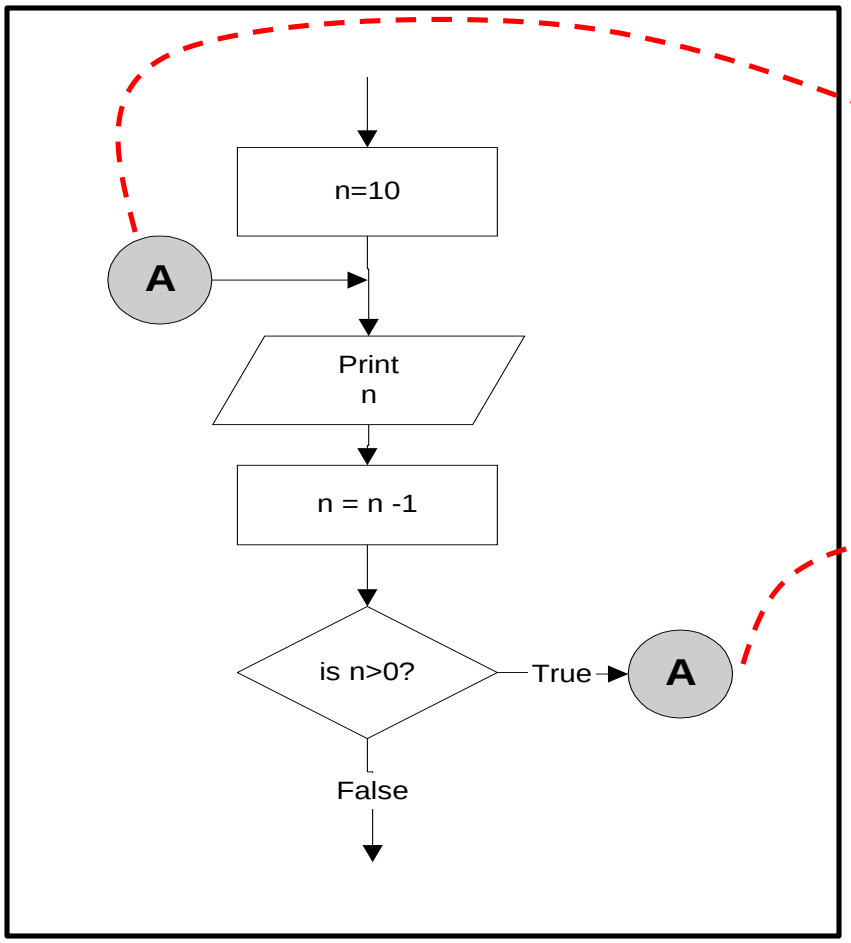
- **Unconditionally transfer control.**
- `goto` may be used for transferring control from one place to another.
- The syntax is:

```
goto identifier;
```

Control is unconditionally transferred to the location of a local label specified by *identifier*. For example,

```
Again:
...
goto Again;
```

goto statement




```
n=10;

A:
   printf("%d ", n);
   n = n -1;

   if (n>0)
         goto A;
```

```
Output:

10 9 8 7 6 5 4 3 2 1
```

Program to show goto statement.

```
#include<stdio.h>
void main()
{
 int x;
 printf("enter a number: ");
 scanf("%d",&x);
 if(x%2==0)
       goto even;
 else
       goto odd;
 even:
   printf(" %d is even", x);
   return;
 odd:
   printf("%d is odd", x);
}
```

```
enter a number: 18
18 is even
```

`return` statement

- **Exits the function.**
- return exits immediately from the currently executing function to the calling routine, optionally returning a value. The syntax is:
- return [*expression*];
- For example,

```
int sqr (int x){
    return (x*x);
 }
```

CSE101-Lec 9

Formatted and Unformatted Input/Output functions

Introduction

- Presentation of output is very important.
- Formatted functions scanf and printf :
  - these functions input data from standard input stream and
  - output data to standard output stream.
- Include the header `#include<stdio.h>`

Standard I/O Functions

- There are many library functions available for standard I/O.
- These functions are divided into two categories:
  - **Unformatted functions**
  - **Formatted functions**

Formatted Functions

- With Formatted functions, input and output is formatted as per our requirement
  - For example, if *different values are to be displayed*, how much field width i.e., how many columns on screen, is to be used, and how much space between two values is to be given. If a value to be displayed is of real type, then how many decimal places to output
- Formatted functions are:
  - printf()
  - scanf()

Formatted output with printf function

- **The printf() function: (Formatted output)**

  printf() is an output function that takes text and values from within the program and sends it out onto the screen.

  In general terms, the printf function is written as:

Syntax

```
printf("format-control-string", arg1,arg2,……..,argN);
```

- The format-control-string can contain:
  - Characters that are simply printed as they are
  - Conversion specifications that begin with a % sign
  - Escape sequences that begin with a \ sign

✓ The arguments can be written as constants, single variable or array names, or more complex expressions.

Example

```
printf("Area of circle is %f units
  \n", area);
```

In this :-

"Area of circle is %f units \n"- is a control string.

area - is a variable whose value will be printed.

%f- is the conversion specifier indicating the type of corresponding value to be printed.

Formatted Functions

The scanf() function: (Formatted input)

scanf() is a function that reads data from the keyboard. It interprets character input to the computer and stores the interpretation in specified variable(s).

In general terms, the scanf function is written as:

Syntax

```
scanf (format-control-string, arg1, arg2,……., argN);
```

- The format-control-string can contain:
  - Describes the format of the input.
  - Conversion specifications that begin with a % sign.
- The arguments are the pointers to variables in which the input will be stored.

Example:

```
scanf("%s %d %f", name, &age,
  &salary);
```

In this :-

"%s %d %f"- is a control string.

name – is a string argument and it's a array name and implicit memory address reference.

age – is a decimal integer variable preceded by &.

salary – is floating-point value preceded by &.

Reading data

Conversion Specifier	Description
d	Read signed decimal integer
i	Read a signed decimal integer
u	Read an unsigned decimal integer
h or l	Used before any integer conversion specifier to indicate that a short or long integer is to be input, respectively
e, E, f, g, G	Read a floating-point value
c	Read a character
s	Read a string
p	Read an address
%	Skip the percent sign(%) in the input

```c
#include <stdio.h>
int main( void )
{  int a, c;
   float f;
   char day[10];
   printf( "Enter integers: " );
   scanf( "%d %u", &a, &c);

   printf( "Enter floating-point numbers:" );
   scanf( "%f", &f);

   printf( "%s", "Enter a string: " );
   scanf( "%8s", day );
}
```

```
Enter integers: -89 23
Enter floating-point numbers:
1.34256
Enter a string:
monday
```

Unformatted functions

- The unformatted functions work only with character data type.
- They do not require format conversion symbol for formatting of data types because they work only with character data type
- Unformatted functions are:
  - getchar() and putchar()
  - getch() and putch()
  - gets() and puts()

Unformatted Functions

- C has three types of I/O functions:
  i. Character I/O
  ii. String I/O
  iii. File I/O

getchar()

- This function reads a character-type data from standard input.
- It reads one character at a time till the user presses the enter key.

Syntax

```
Variable-name = getchar();
```

Example:

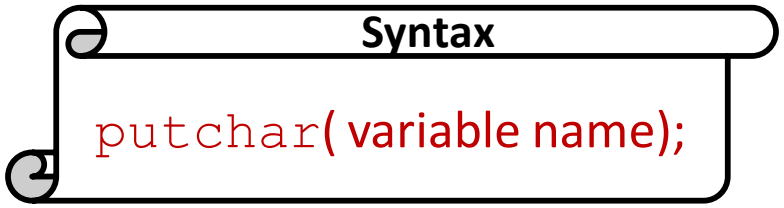
```
char c;
c = getchar();
```

```
#include<stdio.h>
 void main()
{
 char c;
 printf("enter a character");
 c=getchar();
 printf("c = %c ",c);
 }
```

```
Enter a character   k
c = k
```

putchar()

- This function prints one character on the screen at a time which is read by standard input.



Example: char c= ‘c’;

putchar (c);

```
#include<stdio.h>
 void main()
{
char ch;
printf("enter a character: ");
scanf("%c", ch);
putchar(ch);
}
```

```
enter a character: r
r
```

getch() & getche()

- These functions read any alphanumeric character from the standard input device
- The character entered is not displayed by the getch() function until enter is pressed
- The **getche()** accepts and displays the character.
- The **getch()** accepts but does not display the character.

Syntax

```
getche();
```

```
#include<stdio.h>
 void main()
{
    printf("Enter two alphabets:");
    getche();
    getch();
}
```

```
Enter two alphabets a
```

putch()

This function prints any alphanumeric character taken by the standard input device

```
#include<stdio.h>
 void main()
{
      char ch;
      printf("Press any key to continue");
      ch = getch();
      printf(" you pressed:");
      putch(ch);
}
```

```
Press any key to continue
You pressed : e
```

gets()

String I/O

- This function is used for accepting any string until enter key is pressed (string will be covered later)

Syntax

```
char str[length of string in number];
gets(str);
```

```
#include<stdio.h>
 void main()
     {
     char ch[30];
     printf("Enter the string:");
     gets(ch);
     printf("Entered string: %s", ch);
     }
```

```
Enter the string: Use of data!
Entered string: Use of data!
```

puts()

- This function prints the string or character array. It is opposite to gets()

Syntax

```
char str[length of string in number];
gets(str);
puts(str);
```

```
#include<stdio.h>
 void main()
       {
       char ch[30];
       printf("Enter the string:");
       gets(ch);
       puts("Entered string:");
       puts(ch);
       }
```

```
Enter the string: puts is in use
Entered string: puts is in use
```

Mcq

Choose a C unformatted input output function below.

A) gets(), puts()

B) getchar(), putchar()

C) A & B

D) None of the above

MCQ

What is the output of C program.?

```
int main()
 {
char ch='A';
 ch=getchar();
putchar(ch);
 return 0; }//input= S
```

A) A

B) B

C) S

D) Compiler error