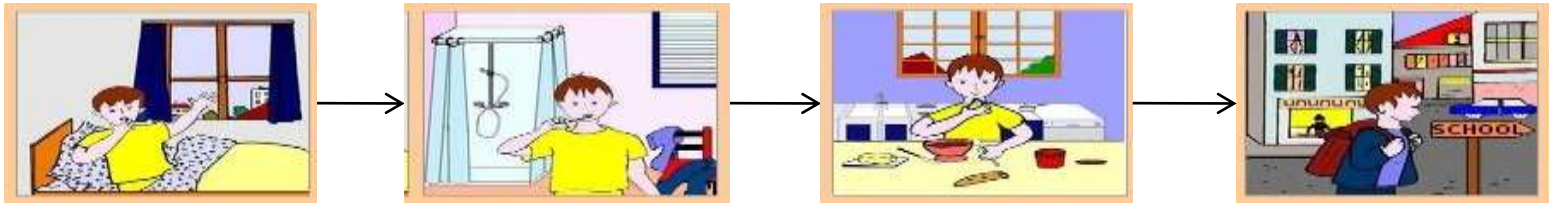
CSE101-Lec#6-Part-1

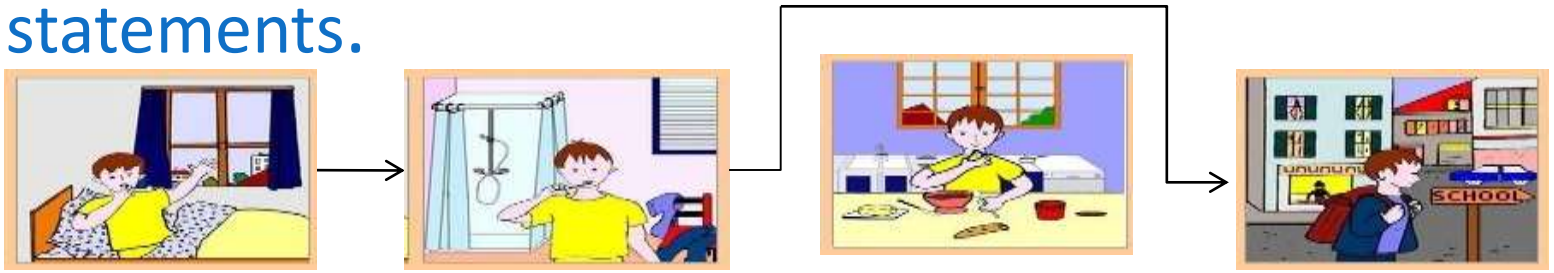
- Control structures(Decision control statements/ or Condition Statements)

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.



1. Wake up;
2. Get ready;
3. If you have enough time, then eat breakfast;
4. Go to school.

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.

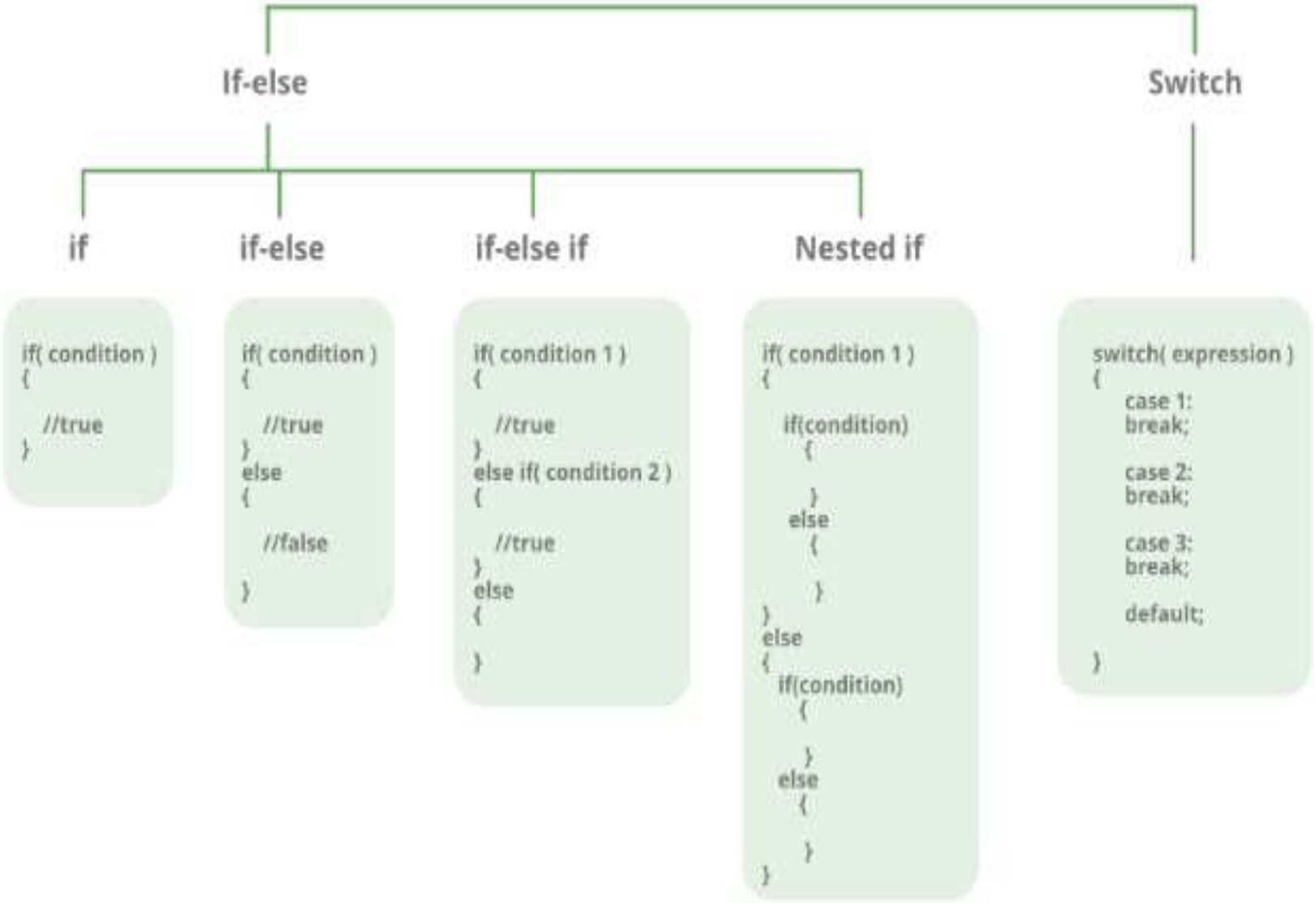
Control Structure

- 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(Condition)
    - `if`, `if…else`, if-else-if, Nested-if ,`switch`
  - Repetition structures (iteration)
    - `while`, `do…while`, `for`

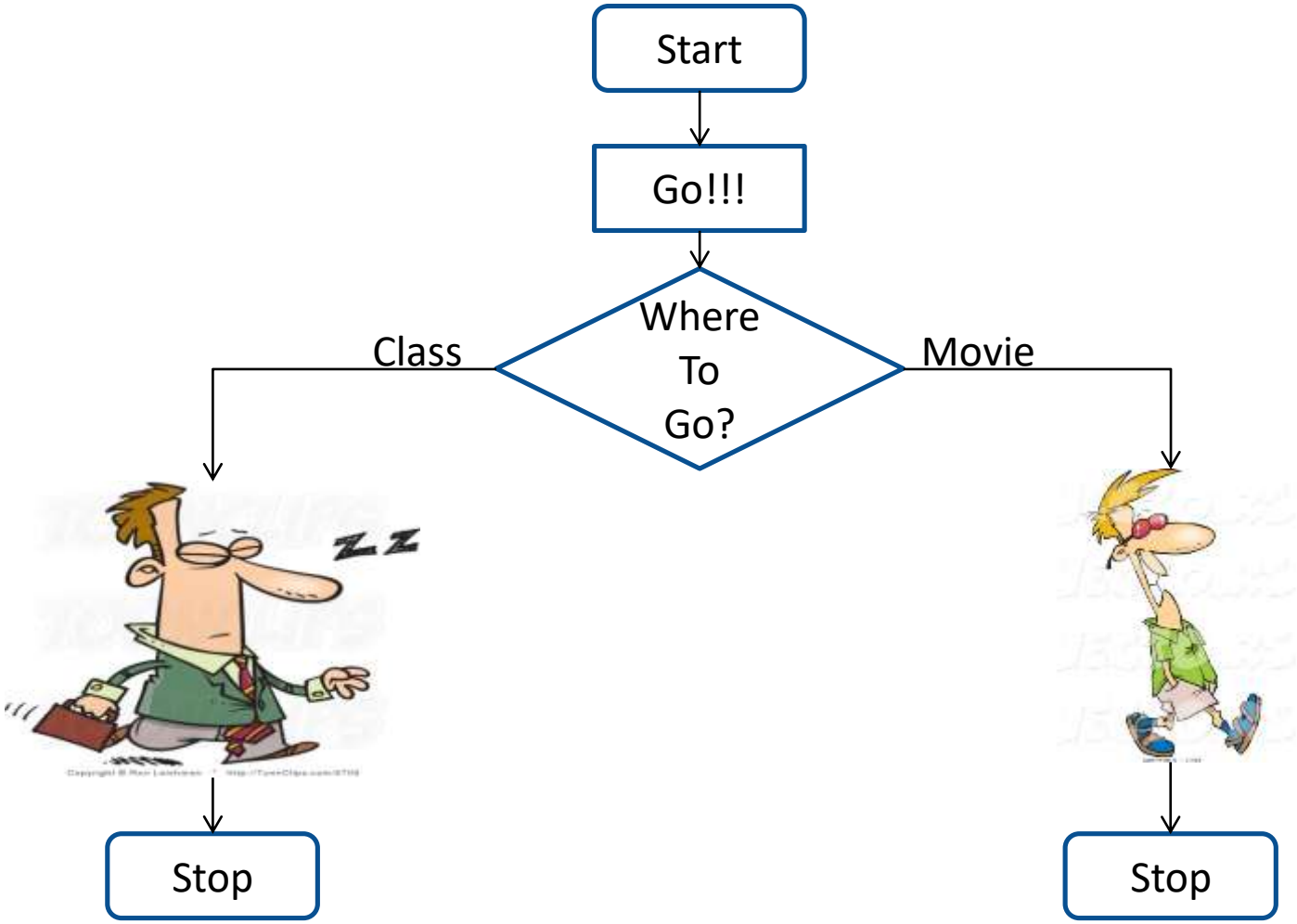
Condition Statements(or Decision control statements or Branching statements)

- 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*
  - *If-else-if statement*
  - *Nested if statement*
  - *switch statement*

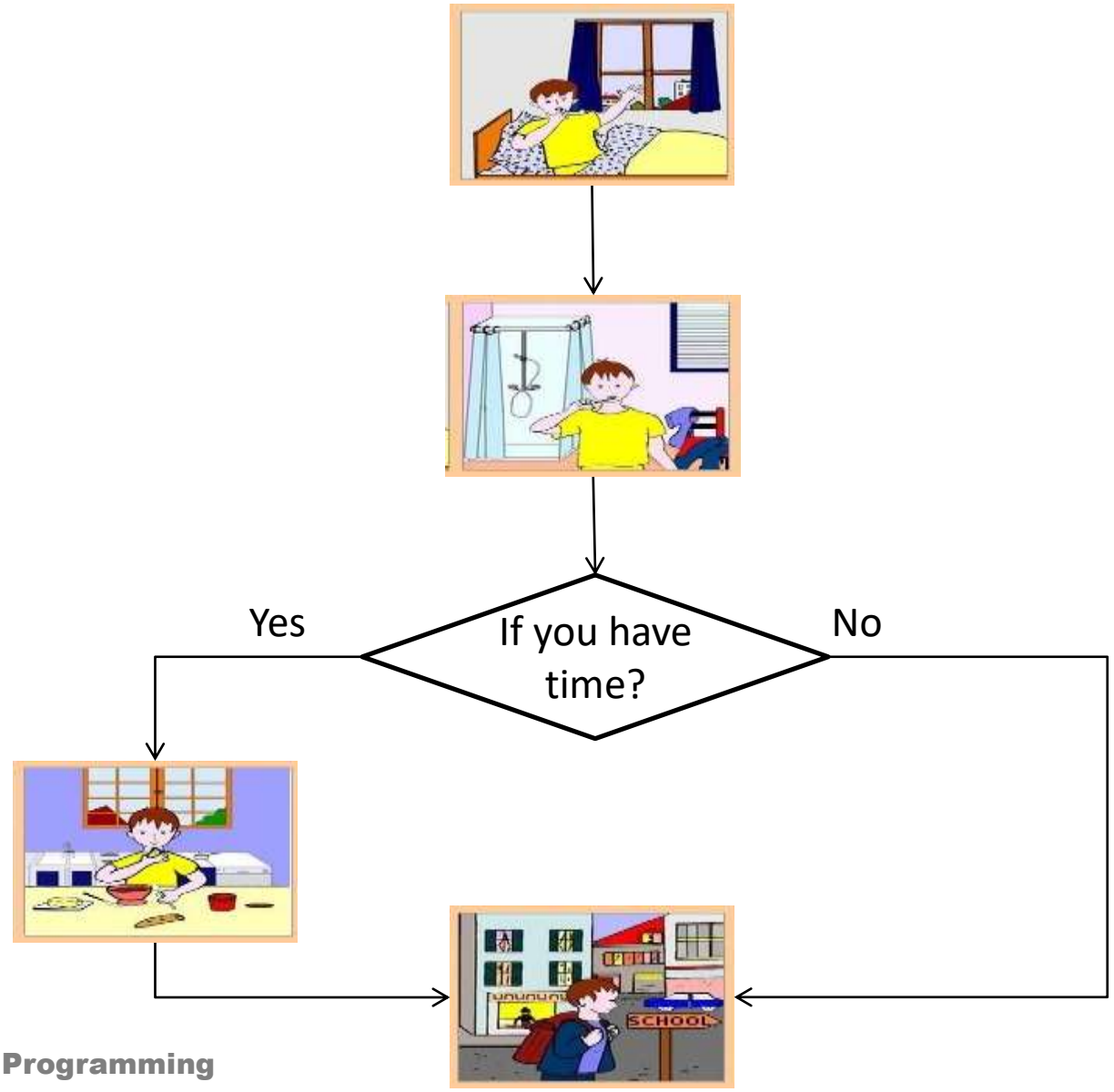
Decision Making



Daily routine



`if` statement



`if` Statement

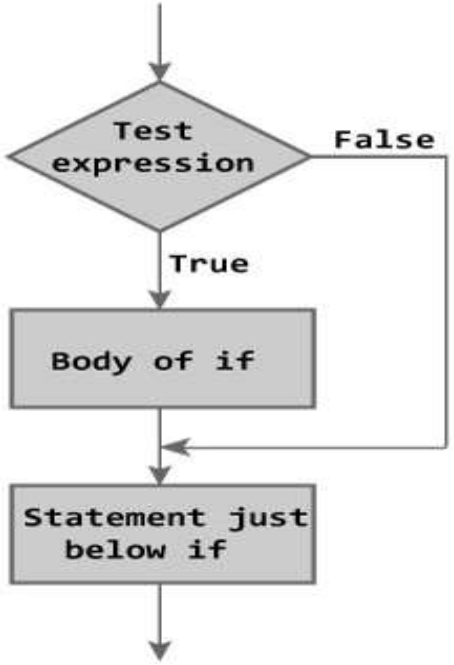
- If statement
  - It is decision making statement uses keyword `if`.
  - It allows the computer to evaluate the expression first
    - and then, depending on whether the value is 'true' or 'false', i.e. non zero or zero it transfers the control to a particular statement.

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:

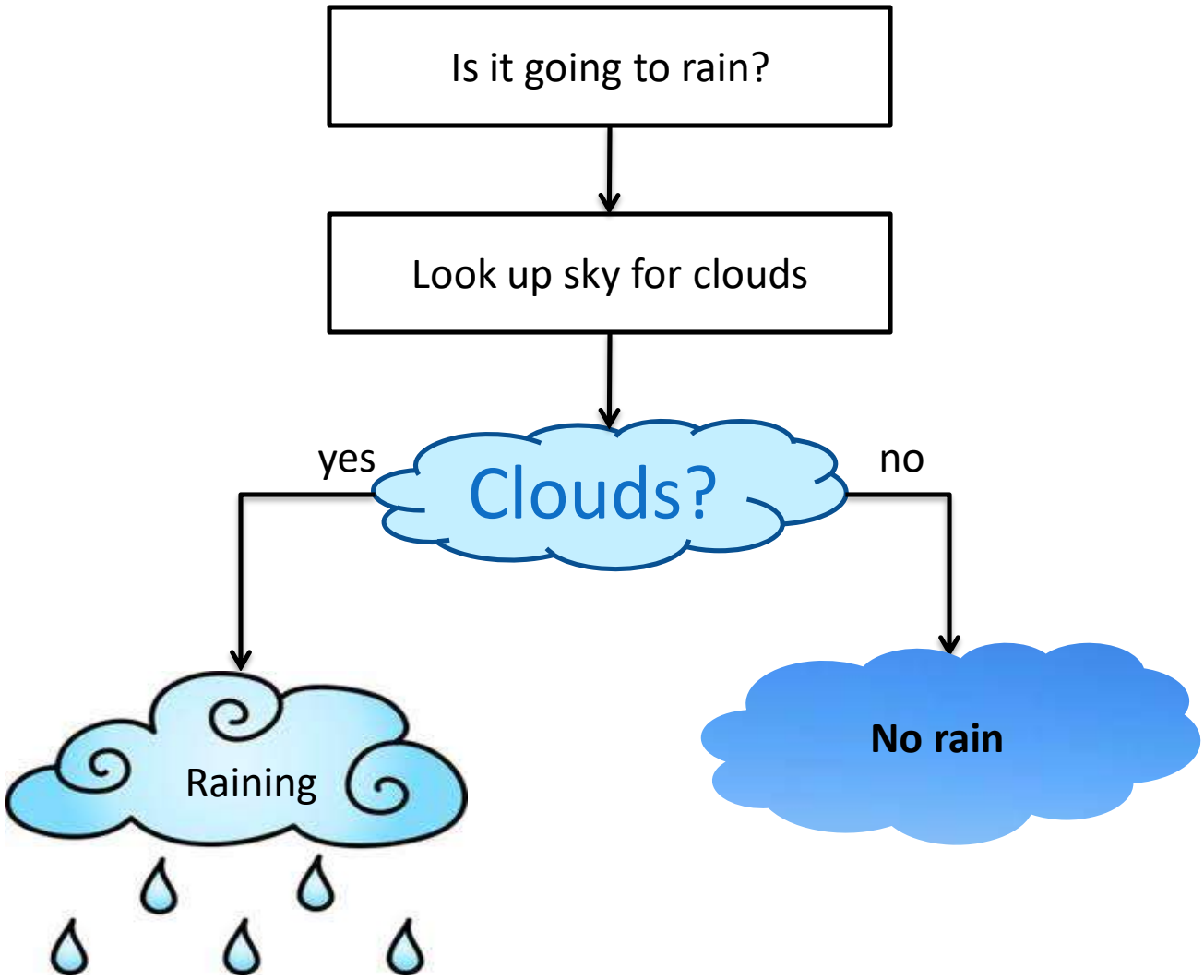
`if` is a C reserved word

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 ???

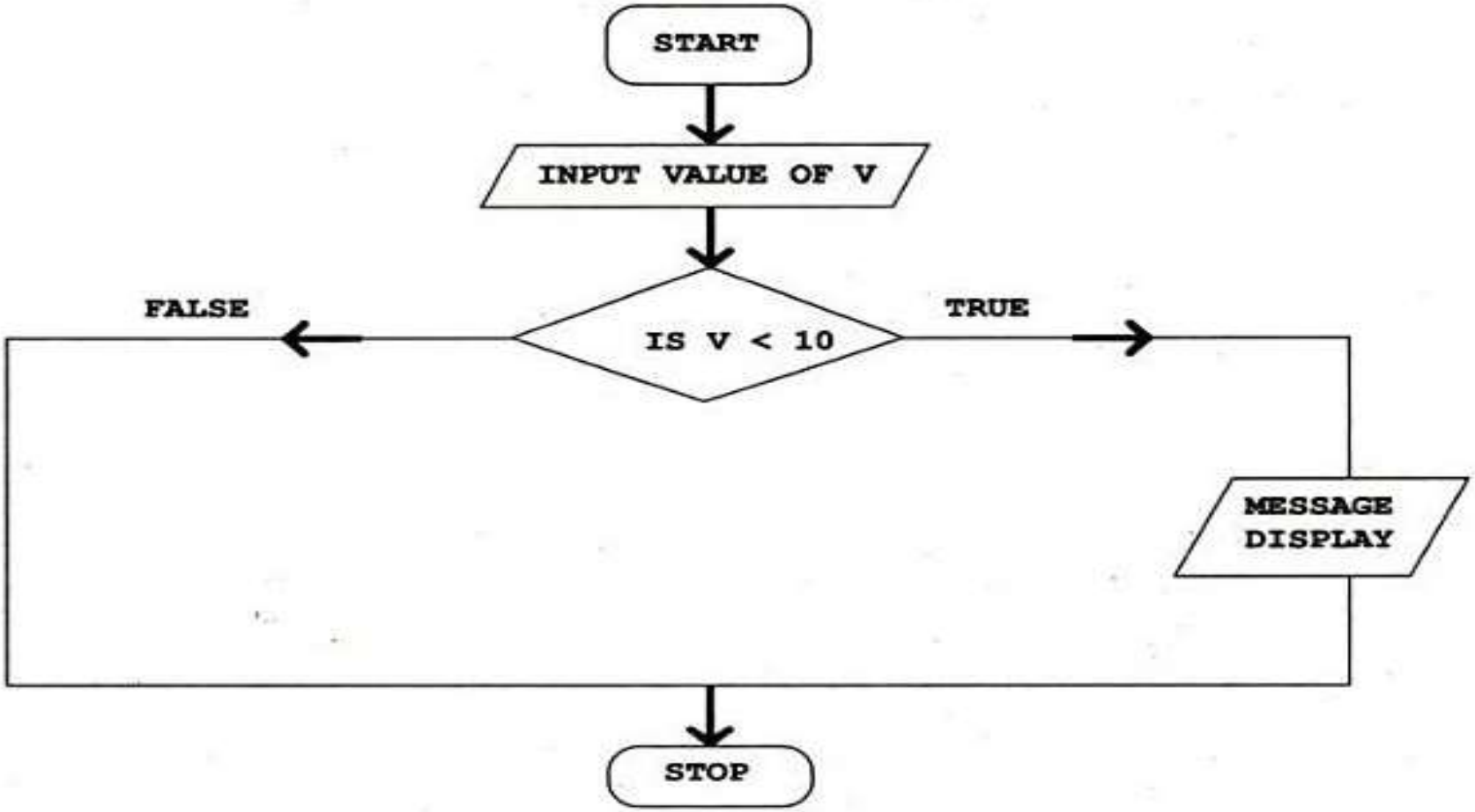

Is it going to rain?
Look up sky for clouds
Clouds?
yes
no
Raining
No rain

```
#include<stdio.h>
int main()
{
 int v;
 printf("Enter the number :");
 scanf("%d", &v);
 if(v<10)
   printf("number is less  than 10");
return 0;
}
```

Program to check whether number is less than 10.

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

Control Flow



`if..else` statement

If you have time?

Yes

No

Grab something to eat along

`if..else` statement

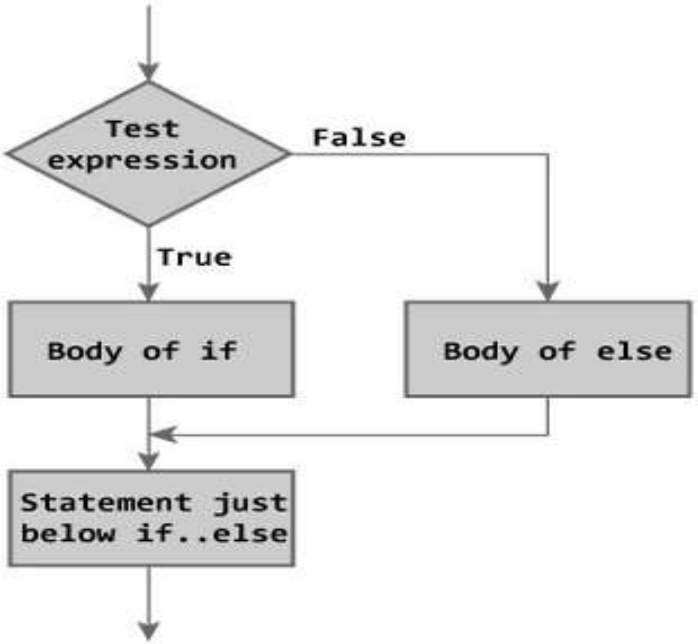
- The `if` statement executes only when the condition following if is true.
- It does nothing when the condition is false.
- The `if..else` statement takes care of the true and false conditions.

`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`.

```
#include<stdio.h>
int 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");
return 0;
}
```

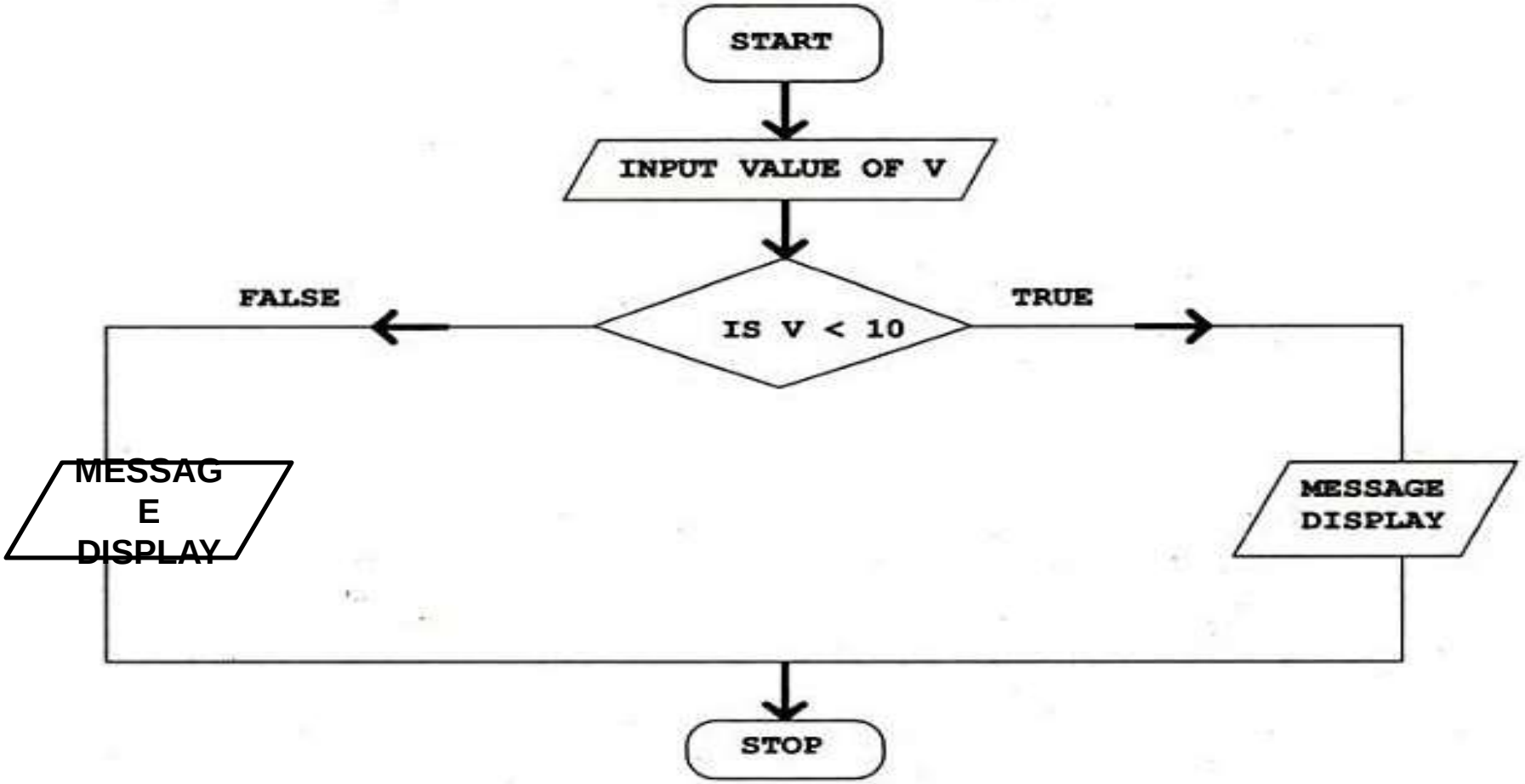
Example : Program to check whether number is less than 10.

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

or

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

Control Flow



Q1

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int x = 5;
    if (x < 1)
        printf("hello");
    if (x == 5)
        printf("hi");
    else
        printf("no");
    return 0;
}
```

A. hi

B. hello

C. no

D. error

Q2

What will be the output of the following C code?

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

A. hi

B. how are you

C. hello

D. hihello

Q3

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int x = 5;
    if (x < 1);
        printf("Hello");

}
```

A. Nothing will be printed

B. Compile time error

C. Hello

D. Logical error

Q4

```
#include<stdio.h>
int main()
{
float x=2.3;
if(x==2.3)
{
printf("Hi");
}
else
{
printf("Hello");
}
return 0;
}
```

A. Hi
B. Hello
C. Compile time error
D. None of these

Q5

```
#include<stdio.h>
int main()
{
int x=-1;
if(x)
{
printf("Hi");
}
else
{
printf("Hello");
}
return 0;
}
```

A. Hi
B. Hello
C. Compile time error
D. None of these

Q6

What is the output of this C code?

```
#include <stdio.h>
int main()
{
    float f = 0.1;
    if (f == 0.1)
        printf("True");
    else
        printf("False");
    return 0;
}
```

A. True

B. False

C. Compile time error

D. None of these

If-else-if

- `if-else-if` statement is used when program requires more than one test expression.
- We can check multiple conditions, and what so ever condition is true, that part will work
- Here, a user can decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions are true, then the final else statement will be executed.

If-else-if ladder

Syntax

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

```
#include<stdio.h>
int 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");
 }
return 0;
}
```

Program to check whether number is less than 10.

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

or

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

Program to print grades of students marks.

```
#include<stdio.h>
int 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");
      }
return 0;
}
```

```
66.70
Grade D
```

or

```
78.00
Grade C
```

Q1

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

A. inside if

B. inside elseif

C. inside if
   inside elseif

D. Compile time error

Q2

What will be the output of the following C code?

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

A. true

B. false

C. Compile time error

D. undefined behaviour

Q3

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int x = 0;
    if (x == 0)
        printf("true, ");
    else if (x = 10)
        printf("false, ");
    printf("%d\n", x);
    return 0;
}
```

A. false, 0

B. true, 0

C. true, 10

D. compile time error

Nested if

- A nested if in C is an if statement that is the target of another if statement. Nested if statements means an if statement inside another if statement. C allows us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

Syntax

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
}
```

Program example

```
// C program to illustrate nested-if statement
#include <stdio.h>

int main()
{
        int i = 10;
        if (i == 10)
        {
        // First if statement
        if (i < 15)
        printf("i is smaller than 15\n");

// Nested - if statement
// Will only be executed if statement above  is true
        if (i < 12)
        printf("i is smaller than 12 too\n");
        else
        printf("i is greater than 15");
        }
        return 0;
}
```

Output

- i is smaller than 15
- i is smaller than 12 too

What will be the output of following code?

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

A. true

B. false

C. Depends on the compiler

D. Nothing will be printed

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");
    return 0;
}
```

A.inside if

B.inside else if

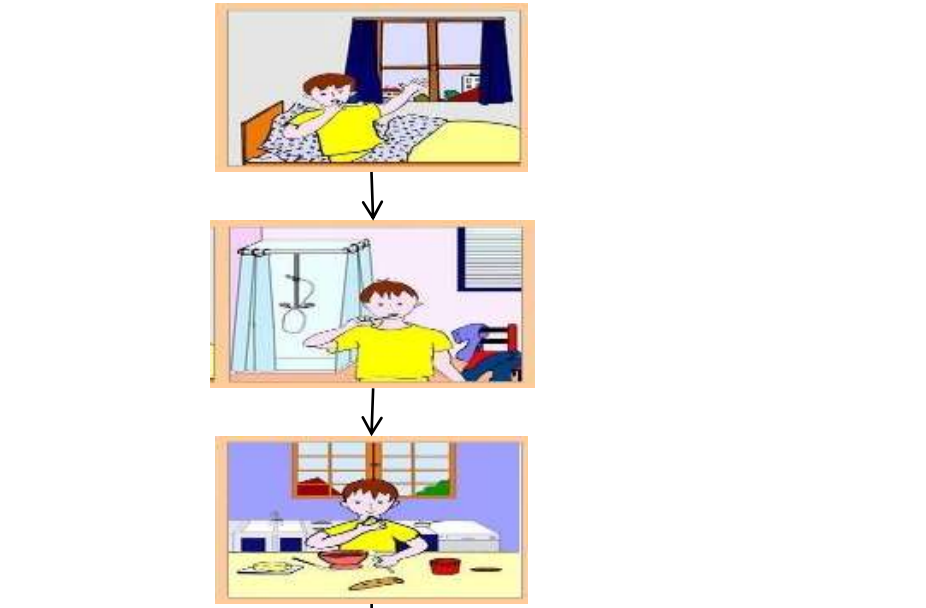
C.inside else

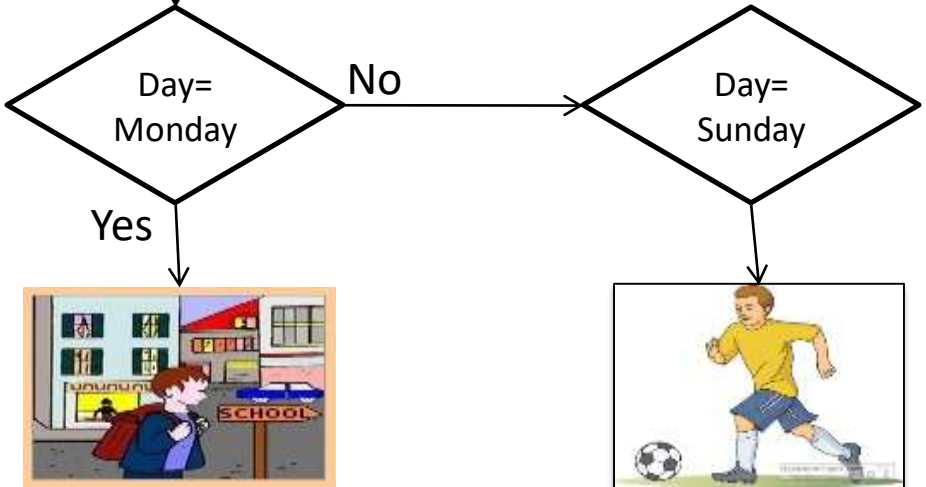
D.Compile time error

`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. Nesting of switch is allowed.
9. **Variables** are not allowed in switch case label..

Syntax error in `switch` statement

```
switch(pt){
      case count:
       printf("%d", count);
       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

Floating point number cannot be used

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

constant expression should be used

Program to show switch statement in geometry

```
#include<stdio.h>
int 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");
     break;
    }
return 0;
}
```

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

Q1

```
#include <stdio.h>
  int main()
  {
     double ch;
     printf("enter a value between 1 to 2:");
     scanf("%lf", &ch);
     switch (ch)
     {
       case 1:
          printf("1");
          break;
       case 2:
          printf("2");
          break;
     }
     return 0;
  }
```

A. Compile time error

B. 1

C. 2

D. Nothing will be displayed

Q2

What will be the output of the following C code? (Assuming that we have entered the value 1 in the standard input)

```
#include <stdio.h>
int main()
{
   int ch;
   printf("enter a value between 1 to 2:");
   scanf("%d", &ch);
   switch (ch)
   {
     case 1:
       printf("1 ");
     default:
       printf("2");
   }
   return 0;
}
```

A. 1

B. 2

C. 1 2

D. Compile time error

Q3

What will be the output of the following C code? (Assuming that we have entered the value 1 in the standard input)

```
#include <stdio.h>
int main()
{
   int ch;
   printf("enter a value between 1 to 2:");
   scanf("%d", &ch);
   switch (ch)
   {
     case 1:
       printf("1 ");
       printf("hi");
       break;
     default:
       printf("2\n");
   }
}
```

A. 1 hi

B. 2

C. hi

D. 1

Q4

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
   int x = 97;
   switch (x)
   {
     case 'a':
       printf("yes ");
       break;
     case 97:
       printf("no");
       break;
   }
}
```

A. yes

B. yes no

C. Duplicate case value error

D. Nothing will be displayed

Q5

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
   int a = 1;
   switch (a)
   {
     case a:
       printf("Case A ");
     default:
       printf("Default");
   }
  return 0;
}
```

A. Output: Case A

B. Output: Default

C. Output: Case A Default

D. Compile time error

CSE101-Lec#8 and 9

- Formatted and Unformatted Input/Output functions
- Type conversion
- Type modifiers

Outline

- Formatted Input/Output functions
  - printf() function
  - scanf() function
- Conversion Specifiers

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()

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()

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.

Printing Integers

- Integer values can be 0, 890, -328.

Conversion Specifier	Description	Example
d	Display as a signed decimal integer.	printf("%d", -890);
i	Display as a signed decimal integer.	printf("%i", -890);
u	Display as an unsigned decimal integer.	printf("%u", 890);
h or l	Used before any integer conversion specifier to indicate that a short or long integer is displayed, respectively	printf("%hd", 890); printf("%ld", 800000000L)

```
#include <stdio.h>
int main( void )
{
   printf( "%d\n", 890);
   printf( "%i\n", 890); // i same as d in printf
   printf( "%d\n", +890 ); // plus sign does not //print
   printf( "%d\n", -890 ); // minus sign prints
   printf( "%hd\n", 32000 );
   printf( "%ld\n", 2000000000L ); // L suffix makes
   //literal a long
   printf( "%u\n", 890 );
   printf( "%u\n", -890 );  // Not allowed
}
```

```
890
890
890
-890
32000
200000000
890
3246435674
```

Printing Floating-Point number

- Decimal point numbers like 0.01, 98.07 or -23.78

Conversion Specifier	Description	Example
e or E	Display a floating-point value in exponential notation.	printf(“%e”,-1234567.89);
f or F	Display floating-point values in fixed-point notation	printf(“%f”,1234567.89);
g or G	Display a floating-point value in either the floating-point from f or the exponential form e based on the magnitude of the value	printf(“%g”, 1234567.89);
L	Used before any floating-point conversion specifier to indicate that a long double is displayed.	printf(“%lf”,1234567.89L);

```
#include <stdio.h>
int main( void )
{
   printf( "%e\n", 1234567.89 );
   printf( "%e\n", -1234567.89 );//minus
  prints
   printf( "%E\n", 1234567.89 );
   printf( "%f\n", 1234567.89 );
   printf( "%g\n", 1234567.89 );
   printf( "%G\n", 1234567.89 );
}
```

```
1.234568e+006
-1.234568e+006
1.234568E+006
1234567.890000
1.234568e+006
1.234568E+006
```

Printing Strings and Characters

Character = ‘A’ and String= “This is string”

Conversion Specifier	Description	Example
c	Display a single character argument.	printf(“%c”, ‘A’);
s	Displays a string and requires a pointer to a character argument.	printf(“%s”, “This is String”);

Conversion Specifier s causes characters to be printed until a terminating null(‘\0’) character is encountered.

```c
#include <stdio.h>
int main( void )
{
   char character = 'A'; // initialize char
   char string[] = "This is a string"; // initialize char
   array

   printf( "%c\n", character );
   printf( "%s\n", "This is a string" );
   printf( "%s\n", string );

}
```

```
A
This is string
This is string
```

Other Conversion Specifier

Pointer holds the address of another variable.

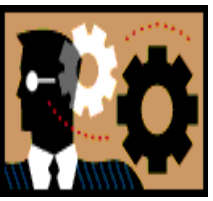
Conversion Specifier	Description	Example

p	Display a pointer value	Int *ptr=&score;
		printf("%p", ptr);
		printf("%p", &score);

%	Displays the percent character.	printf("a%%");

```c
#include <stdio.h>
int main( void )
{
   int *ptr; // define pointer to int
   int x = 12345; // initialize int x

   ptr = &x; // assign address of x to ptr
   printf( "The value of ptr is %p\n", ptr );
   printf( "The address of x is %p\n\n", &x );

   printf( "Printing a %% in a format control string\n" );
}
```

```
The value of ptr is 002ER443
The address of x is 002ER443

Printing a % in a format control string
```



How?

- Till now we have displayed numbers in left justified manner
- Consider the program that displays

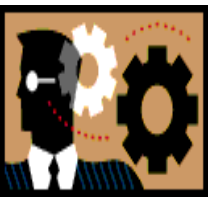
```
1
12
123
1234
12345
```

Printing with Field widths

- Field width: the exact size of field in which data is printed is specified by field width.
- The data is printed in the specified field and **right justified**.
- The integer representing the width size is inserted between percent sign(%) and the conversion specifier(e.g. %8d).

```c
#include <stdio.h>

int main()
{
  printf( "%4d\n", 123 );
   printf( "%4d\n", 1234 );
   printf( "%4d\n\n", 12345 );

  }
```

```
 123
1234
12345

-
```



How?

- Dividing 7 by 3

Answer :

2.33333333333……

But the required output is

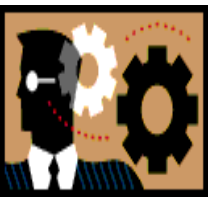
2.3333

Printing with Precision

- Specifies precision with which data is printed.
- Precision with **integer** conversion specifier indicates **the minimum number of digits to be printed.**
- Precision with **floating-point** conversion specifier indicates **the number of digits to appear after the decimal point.**
- Precision with **string** specifier indicates **the maximum number of characters to be written from the string.**

```c
#include <stdio.h>
int main( void )
{
   int i = 873; // initialize int i
   double f = 123.94536; // initialize double f
   char s[] = "Happy Birthday"; // initialize char array s

   printf( "Using precision for integers" );
   printf( "%.4d \n %.9d \n\n", i, i );

   printf( "Using precision for floating-point numbers" );
   printf( "%.3f \n %.3e\n %.3g \n\n", f, f, f );

   printf( "Using precision for strings" );
   printf( "%.11s \n", s );
}
```

```
Using precision for integers
0873
000000873

Using precision for floating-point numbers
123.945
1.239e+002
124

Using precision for strings
Happy Birth
```



How?

Suppose the output required is

```
First\one
there "
There's
```

Printing literals and escape sequences

Escape sequence	Description
\’	Output the single quote(‘) character
\”	Output the double quote(“) character
\\	Output the backslash (\) character
\a	Cause an audible(bell)
\b	Move the cursor back one position on the current line
\n	Move the cursor to the beginning of the next line
\t	Move the cursor to the next horizontal tab position

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
```

Outline

- Unformatted Input/Output functions
  - getchar()
  - putchar()
  - getch()
  - putch()
  - gets()
  - 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:

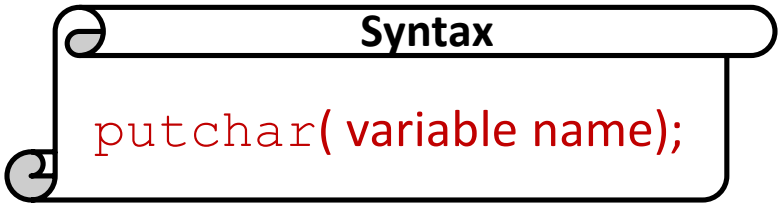
```
char c;
c = getchar();
```

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

Q1

```
#include<stdio.h>
int main()
{
float x=12.6784;
printf("%.3f",x);
return 0;
}
```

A. 12.678
B. 12.6
C. 12.679
D. 12.0

Q2

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=3.456;
printf("%lf",floor(x));
return 0;
}
```

A. 3.460000
B. 3.000000
C. 4.000000
D. 3.500000

Q3

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=3.001;
printf("%lf",ceil(x));
return 0;
}
```

A. 3.010000
B. 3.000000
C. 4.000000
D. 3.500000

Q4

```
#include<stdio.h>
#include<math.h>
int main()
{
double x=10.0,y=7.0;
printf("%lf",fmod(x,y));
return 0;
}
```

A. 1.000000
B. 3.000000
C. 1.428571
D. Error

Q5

```
#include<stdio.h>
#include<math.h>
int main()
{
int x;
x=printf("ABC");
printf(" %d",x);
return 0;
}
```

A. ABC
B. ABC 1
C. 3 ABC
D. ABC 3

Q6

Which of the following is a non-standard input unformatted function in C?

A. printf()
B. getch()
C. getchar()
D. scanf()

Q7

Which of the following unformatted function waits for the user to press the enter key, after the character input is provided?

A. putch()

B. getch()

C. getchar()

D. getche()

Type conversion

- Conversion from one type to another
- It can happen in two ways:

1) Implicit type conversion
2) Explicit type conversion

Implicit type conversion(or automatic type conversion)

- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.
- char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double
- It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Example of implicit conversion

```
// An example of implicit conversion
#include<stdio.h>
int main()
{
        int x = 10; // integer x
        char y = 'a'; // character c

        // y implicitly converted to int. ASCII
        // value of 'a' is 97
        x = x + y;

        // x is implicitly converted to float
        float z = x + 1.0;

        printf("x = %d, z = %f", x, z);
        return 0;
}
```

Output:

x = 107, z = 108.000000

Explicit type conversion

- This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.
- The syntax in C:

(type) expression

- Type indicated the data type to which the final result is converted.

```
// C program to demonstrate explicit type casting
#include<stdio.h>
int main()
{
        double x = 1.2;

        // Explicit conversion from double to int
        int sum = (int)x + 1;

        printf("sum = %d", sum);

        return 0;
}
```

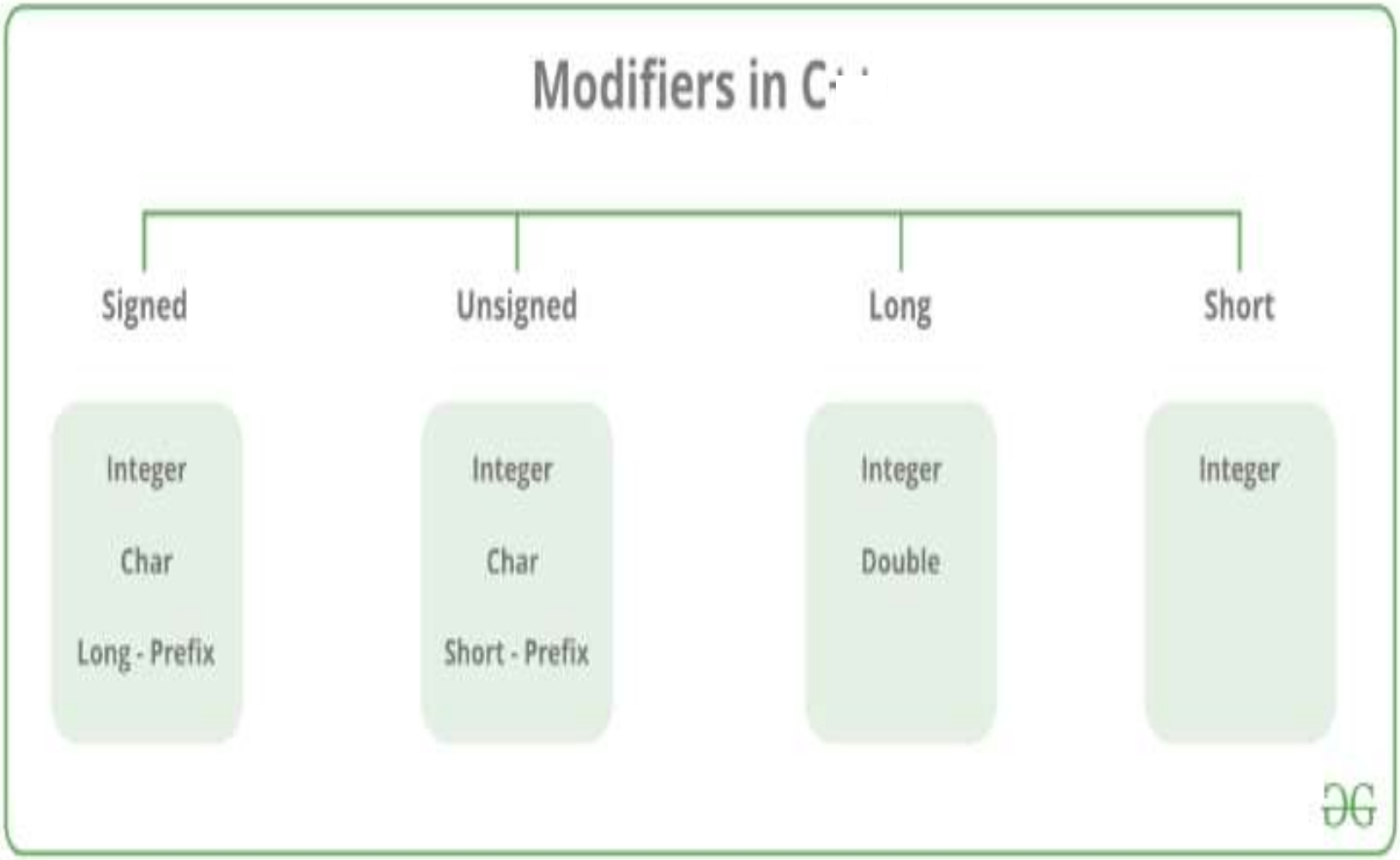
Output:

sum=2

Type modifiers

- Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
- short, long, signed and unsigned are the type modifiers in C

Type modifiers



CSE101-Lec 7

Control Structures

(Repetition structure)

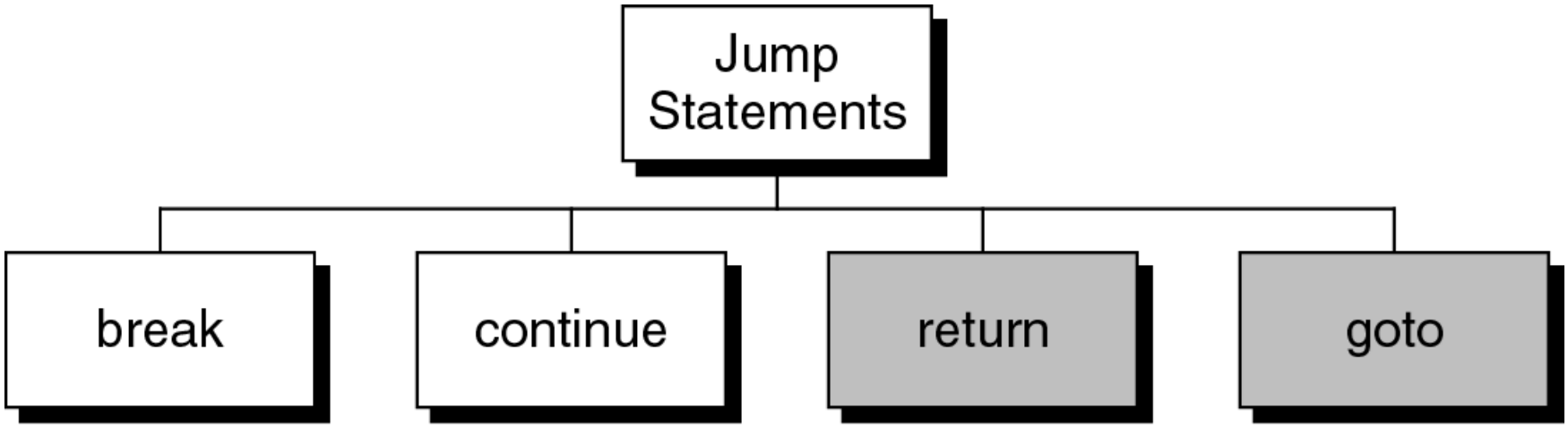
Jump Statements

Outline

- Jump Statements
  - break
  - continue
  - goto
  - return

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.

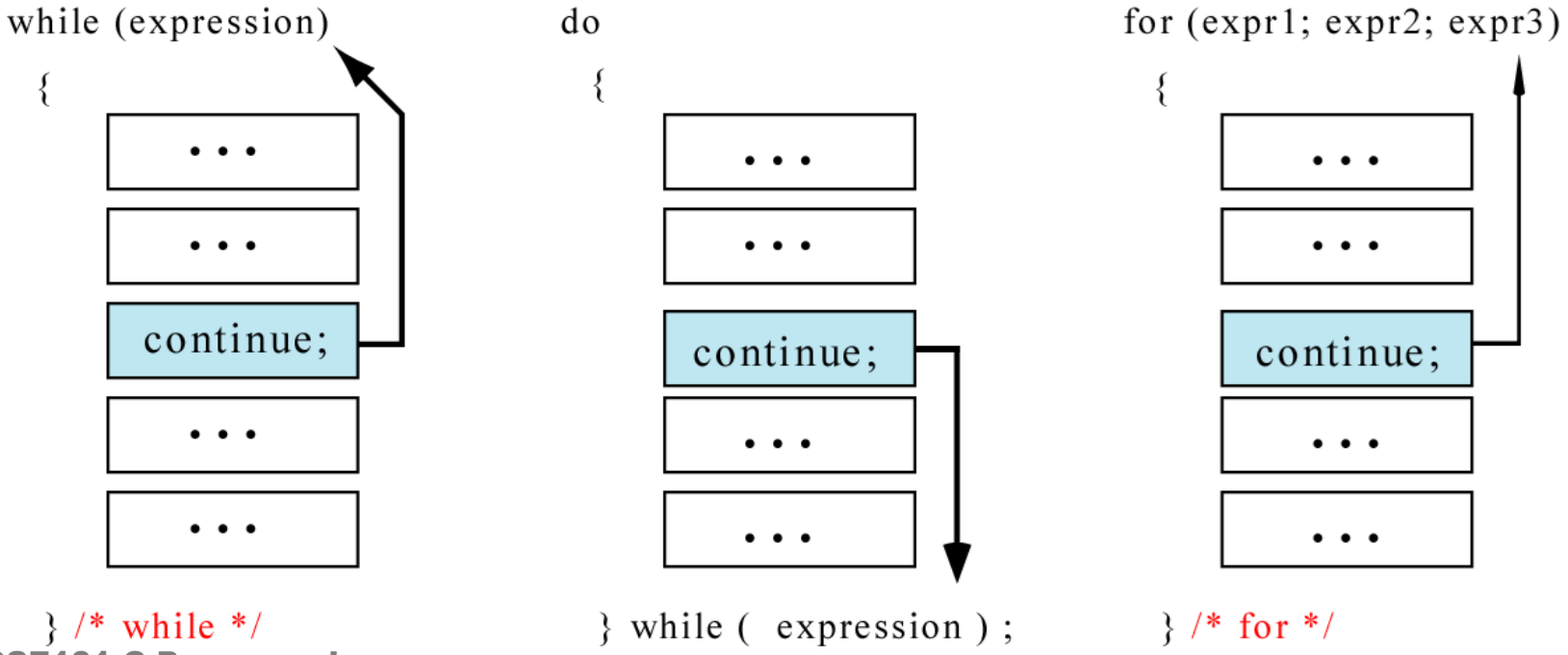
```
10 9 8
```

`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

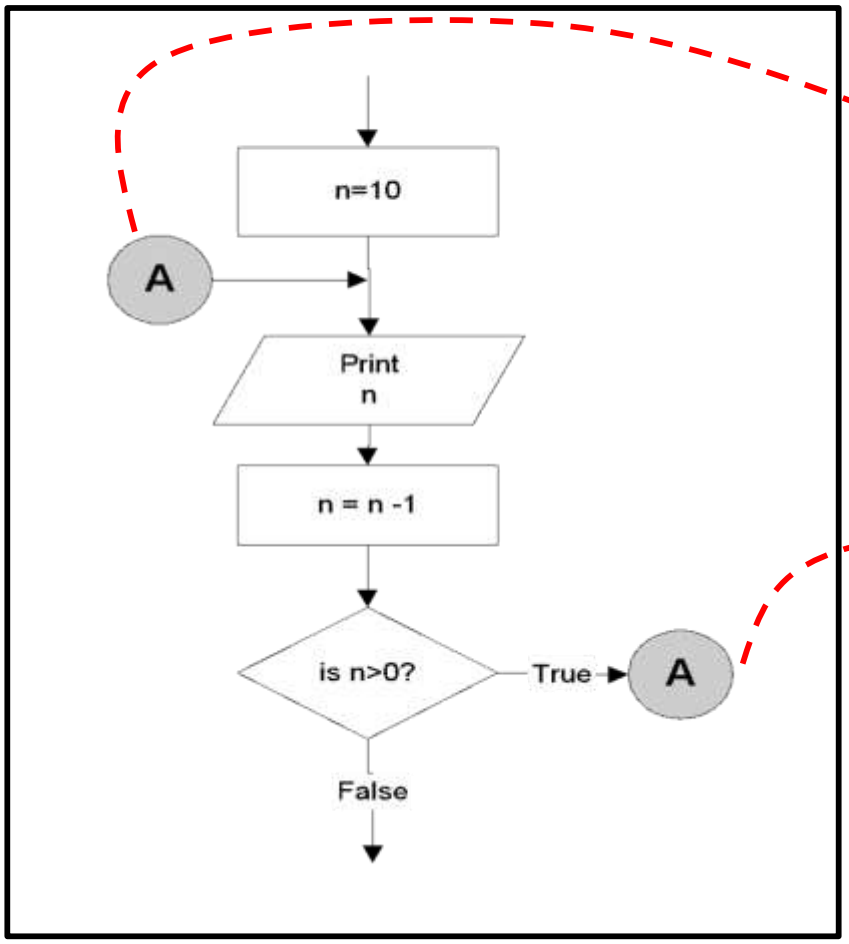
- **Unconditionally transfer control.**
- `goto` may be used for transferring control from one place to another.
- The syntax is:

    `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);
 }
```

Question 1

The continue statement cannot be used with

A. for

B. while

C. do while

D. switch

Answer 1

The continue statement cannot be used with

A. for

B. while

C. do while

D. switch

Question 2

Which keyword can be used for coming out of recursion?

A. return

B. break

C. exit

D. both A and B

Answer 2

Which keyword can be used for coming out of recursion?

A. return

B. break

C. exit

D. both A and B

Question 3

Switch statement accepts.

A. int
B. char
C. long
D. All of the above

Answer 3

Switch statement accepts.

A. int
B. char
C. long
D. All of the above

Question 4

Which loop is guaranteed to execute at least one time.

A. for

B. while

C. do while

D. None of the above

Answer 4

Which loop is guaranteed to execute at least one time.

A. for

B. while

C. do while

D. None of the above

Question 5

A labeled statement consist of an identifier followed by

A. ;

B. :

C. ,

D. =

Question 5

A labeled statement consist of an identifier followed by

A. ;

B. :

C. ,

D. =

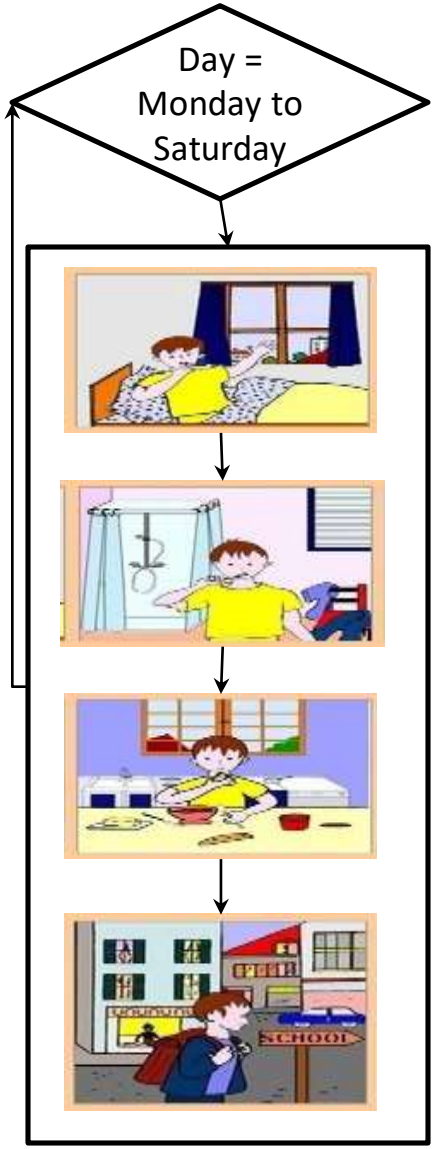
CSE101-Lec#6-Part-2

- Control Structures[Repetition structures/ or Looping statements/ or Iterative statements]

Outline

- Repetition structure/Control Loop Statements
  - for statement
  - while statement
  - do-while statement

Repetition(Going to School)



Looping (repetition)

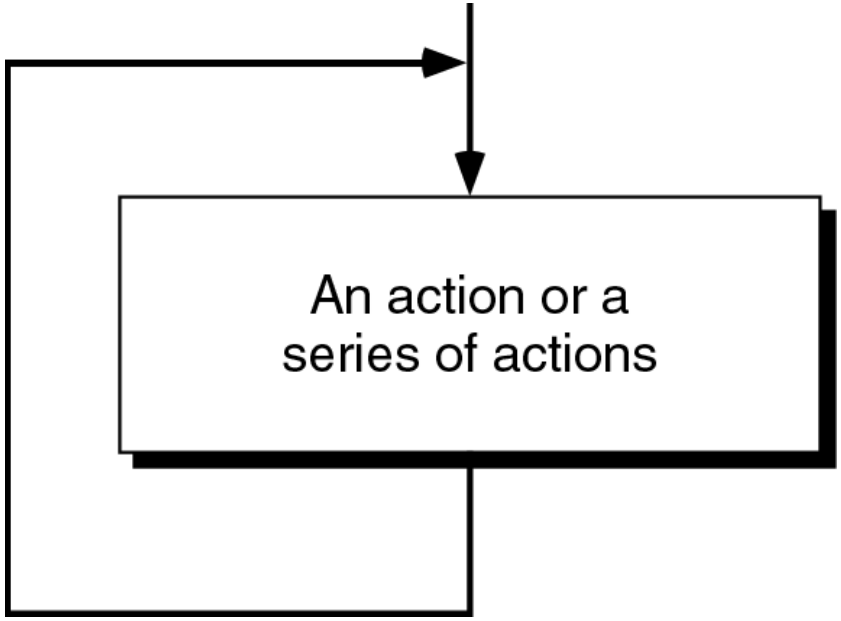
- *What if we want to display hello 500 times?*
  - Should we write 500 printf statements or equivalent ?

➢ Obviously not.

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

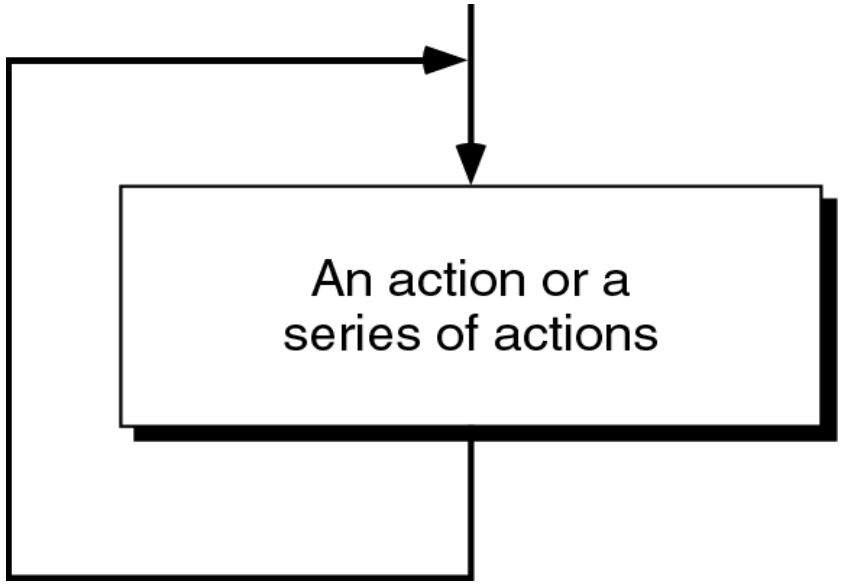
Loop

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



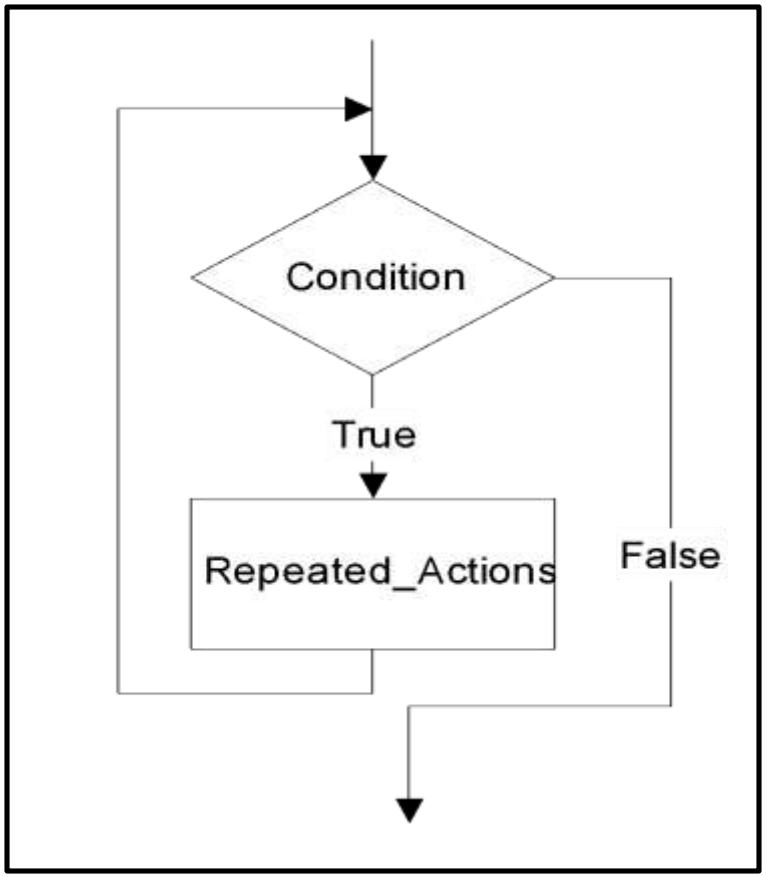
The concept of a loop without condition

- 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.



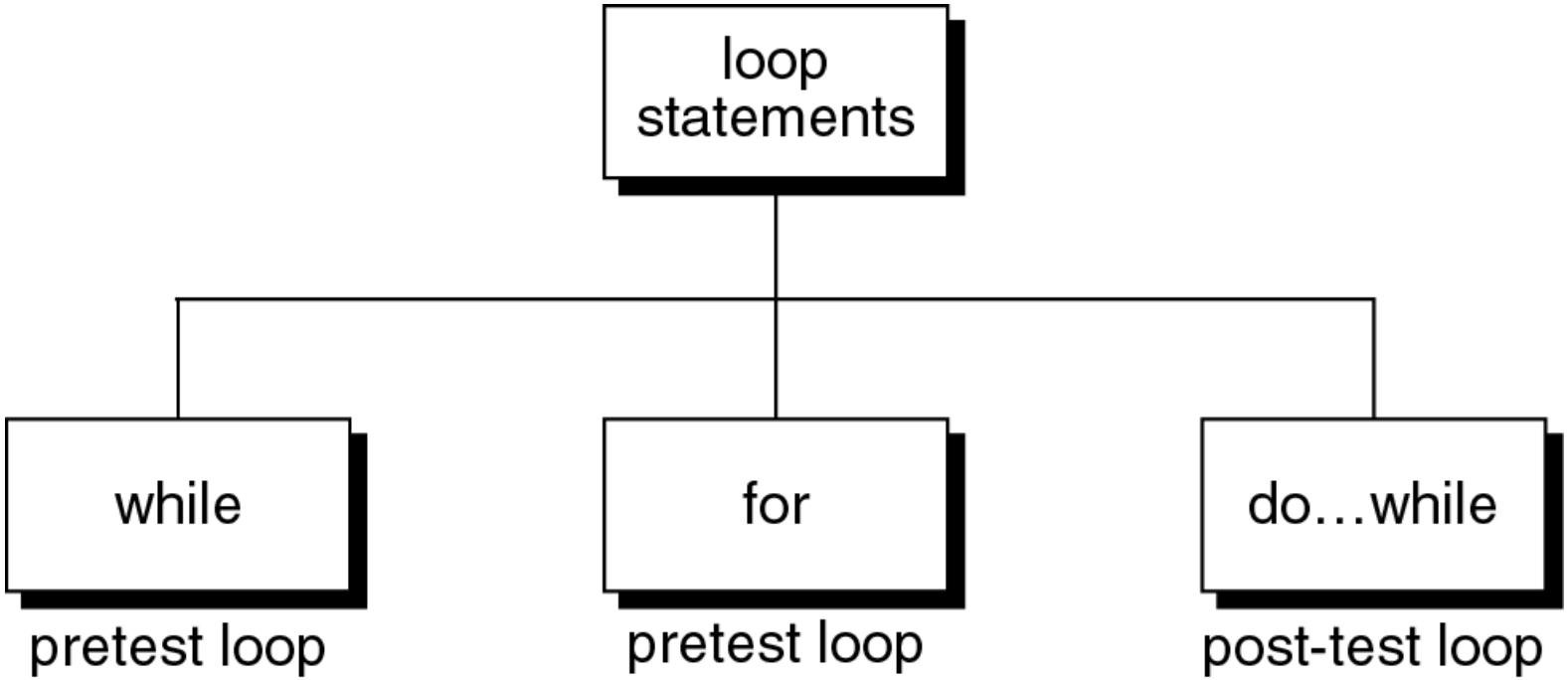
Loop

- 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.

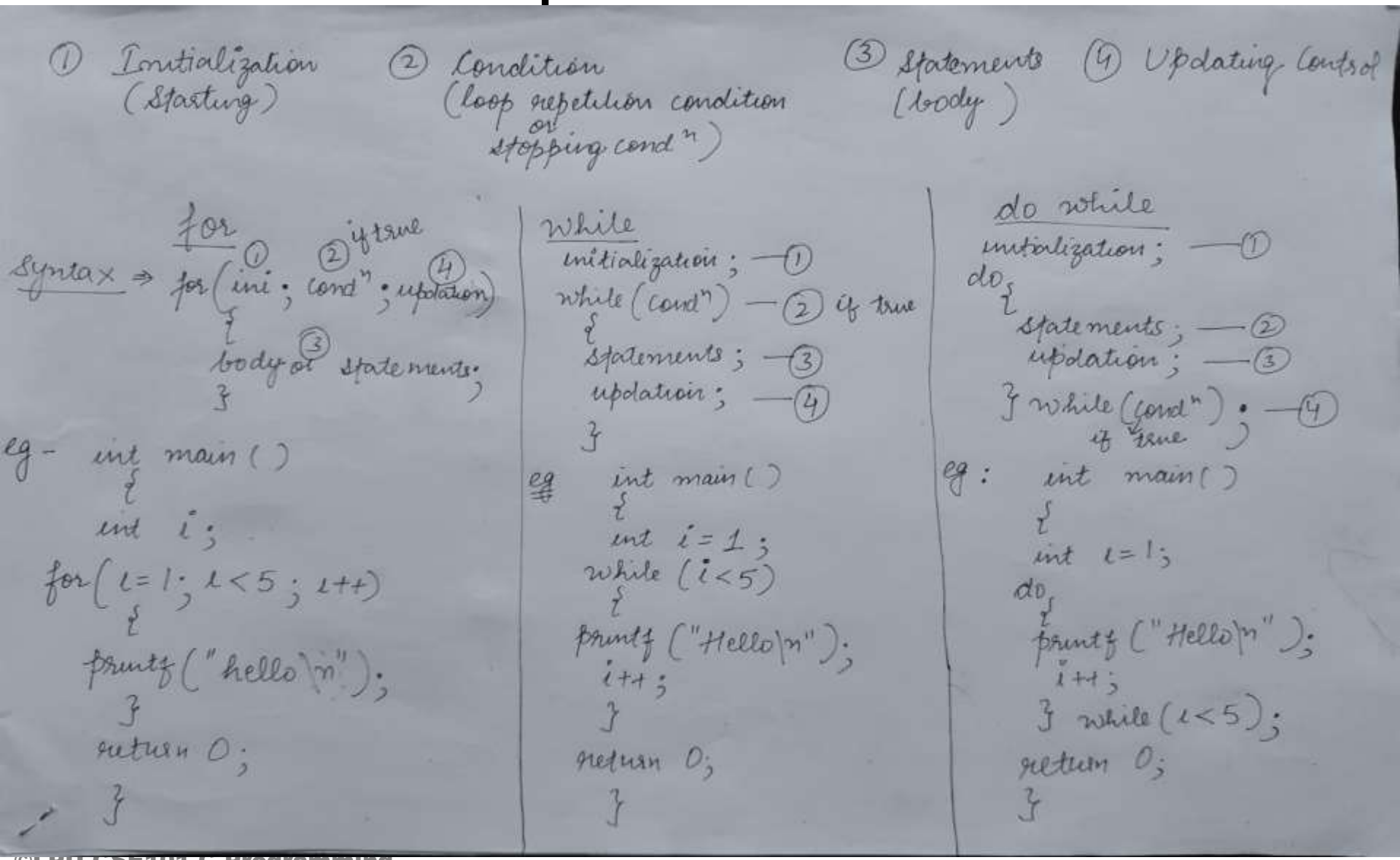


Loop statements

- C provides three loop statements:



Loop statements

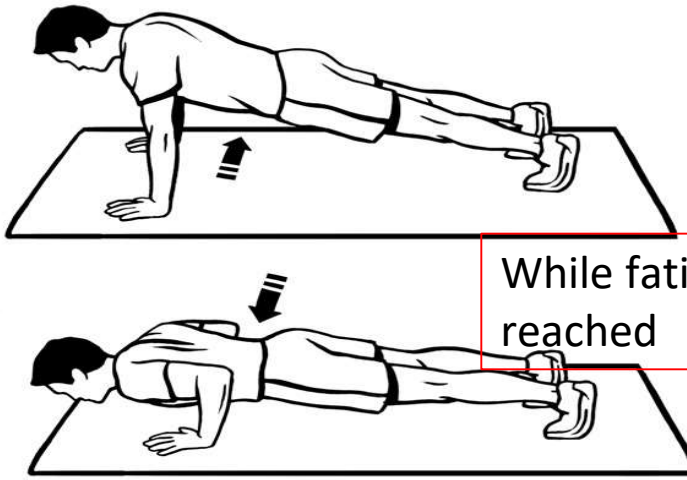


The "while" Statement in C

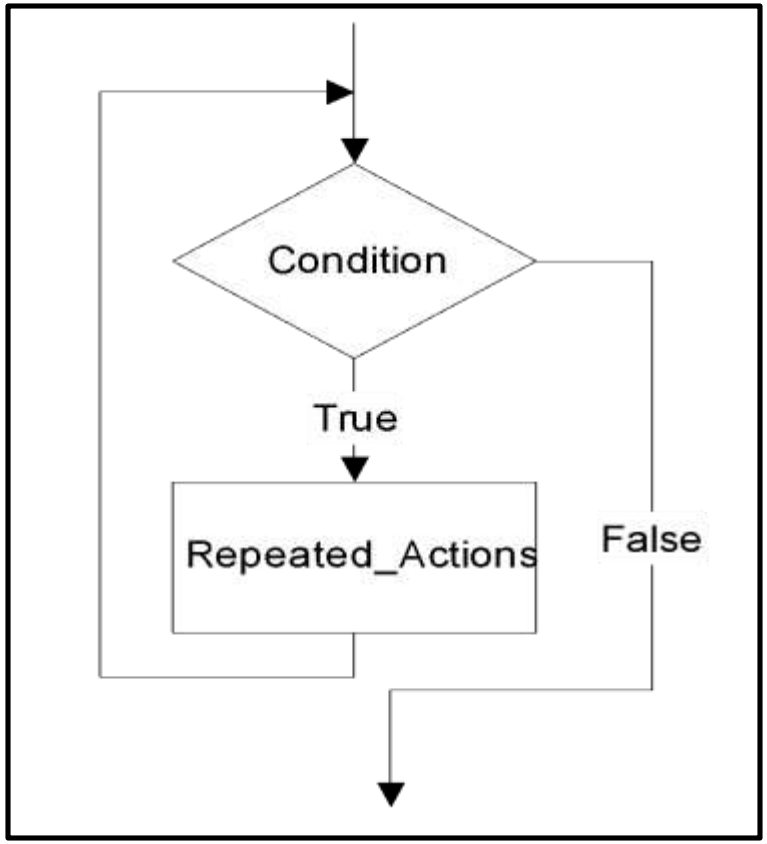
- The syntax of **while** statement in C:

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.

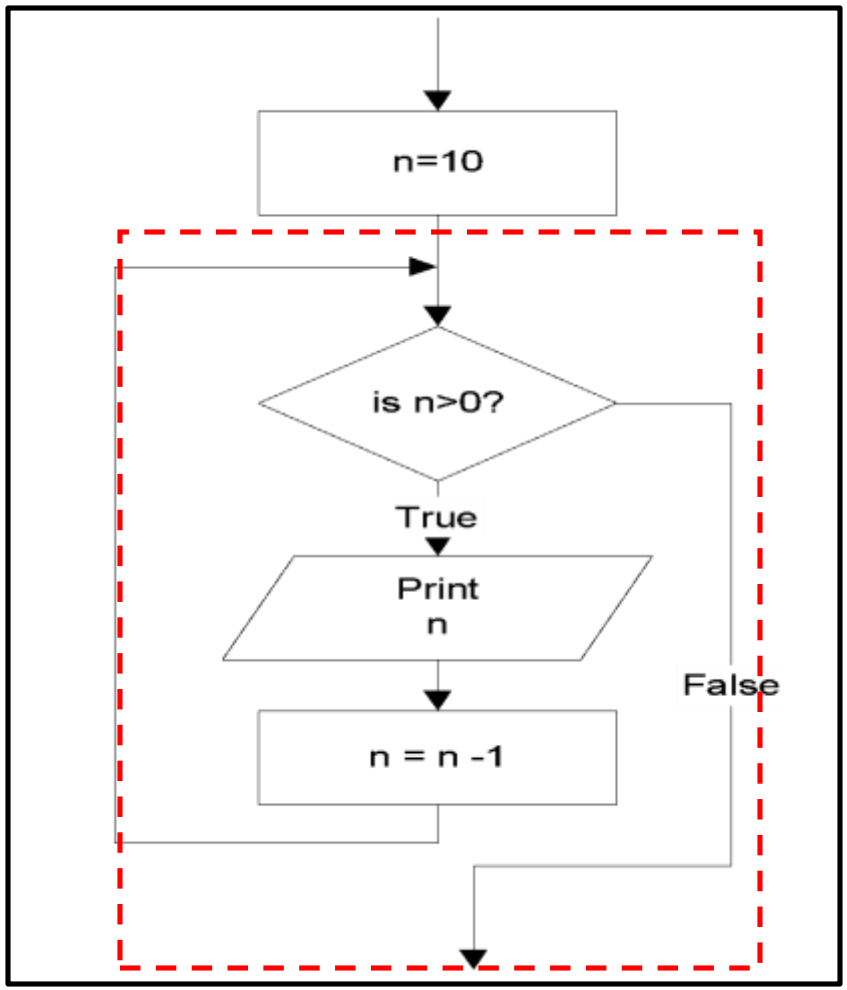
- The ***statement*** is repeated as long as the loop repetition condition is **true**.
- A loop is called an **infinite loop** if the loop repetition condition is always true.
- while loop is known as entry controlled loop, as condition is checked at the beginning/ or entry point

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;
 }
return 0;
}
```

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



count condition

Q1

How many times i value is checked in the following C code?

```
#include <stdio.h>
int main()
{
    int i = 0;
    while (i < 3)
        i++;
    printf("In while loop\n");
}
```

A. 2

B. 3

C. 4

D. 1

Q2

What will be the output of the following C code?

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

A. H

B. H is printed infinite times

C. Compile time error

D. Nothing will be printed

The for Statement in C

- The syntax of `for` statement in C:

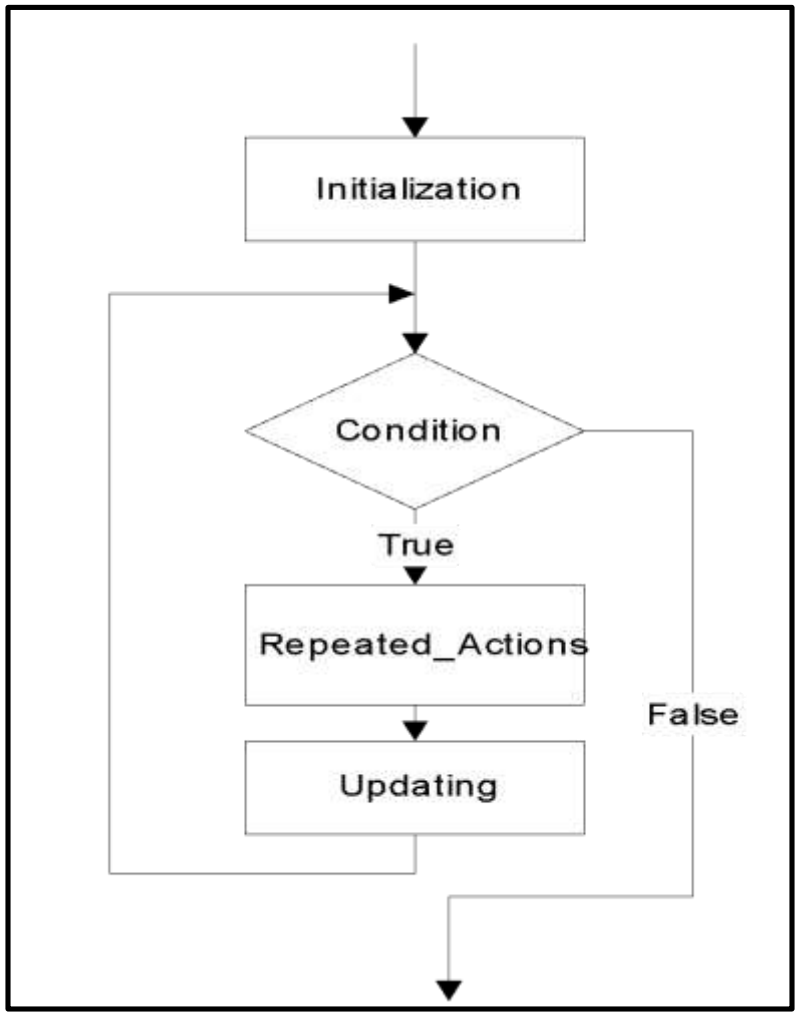
Syntax

```
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.
- It is also known as entry controlled loop as condition is checked first and then loop body executes

for statement

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

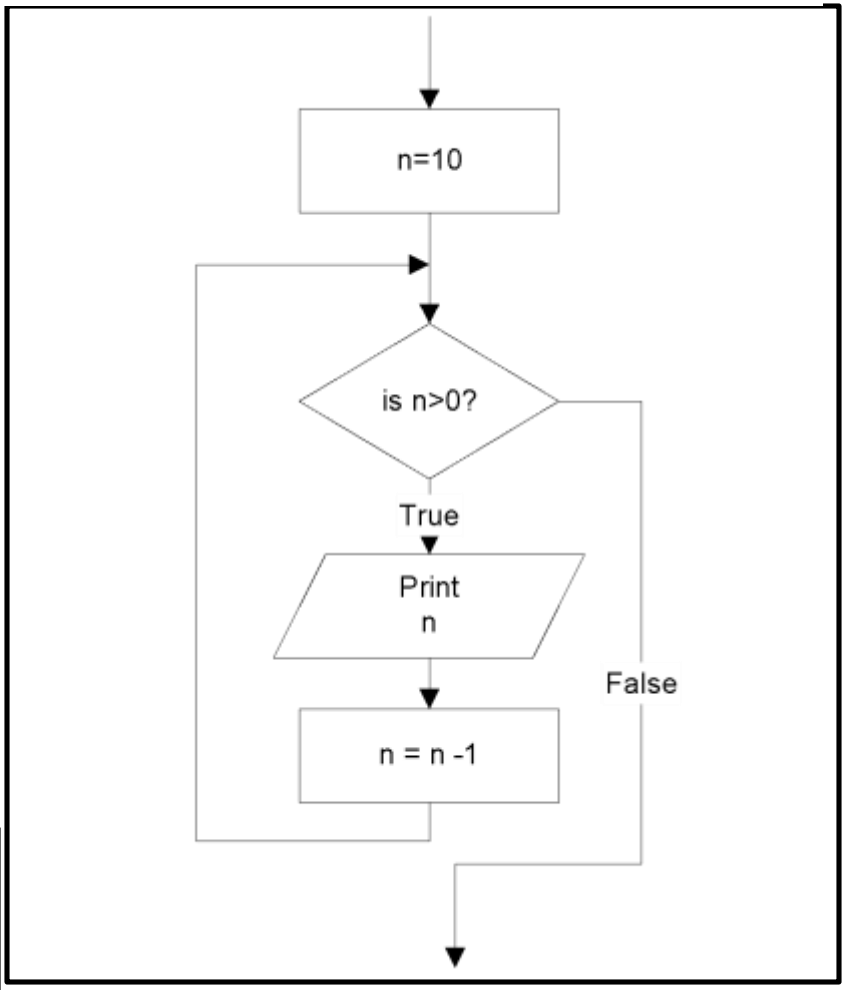


for statement

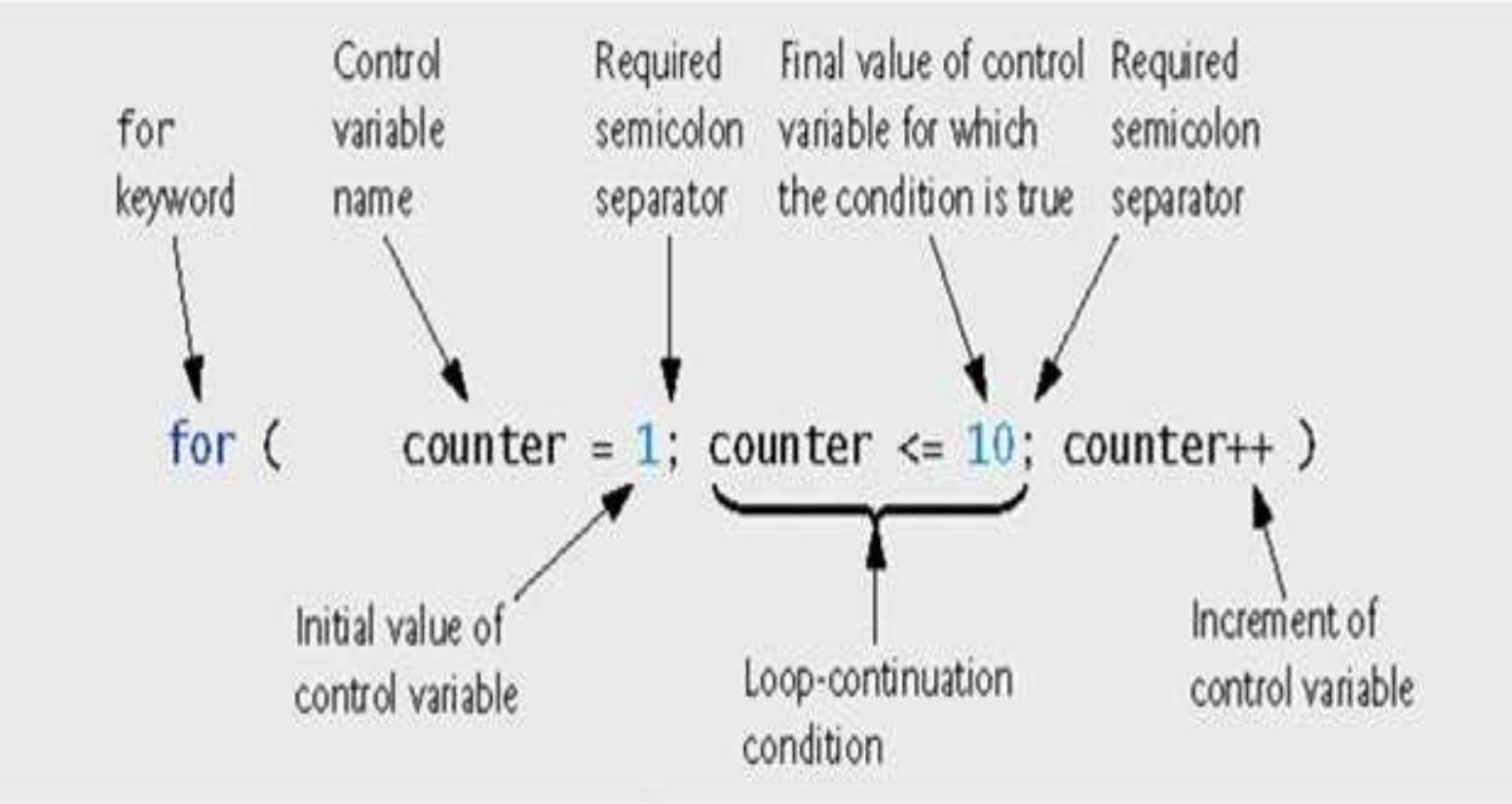
Example: This for statement prints numbers 10 down to 1

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

```
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

Program to find sum of first n numbers

```
#include<stdio.h>
int main()
{
    int n,i,sum=0;
    printf("enter the value");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
    sum=sum+i;
    }
    printf("the sum=%d",sum);
    return 0;
}
```

Program to find sum of digits of a number

```
#include<stdio.h>
int main()
{
    int no,sum=0,n;
    printf("enter the no");
    scanf("%d",&no);
    while(no!=0)
    {
            n=no%10;
            sum=sum+n;
            no=no/10;

        }
    printf("the sum of digits=%d",sum);
    }
```

Q1

```
#include <stdio.h>
int main()
{
  int i;
  for (i = 1; i != 10; i += 2)
    printf("Hello");
  return 0;
}
```

A. Hello will be displayed 5 times

B. Hello will be displayed 4 times

C. Hello will be displayed infinite no. of times

D. Hello will be displayed 6 times

Q2

What will be the output of following code

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

A. Numbers from 1 to 9 will be printed
B. 10
C. 9
D. Infinite loop

for vs while loop

FOR LOOP	WHILE LOOP
Initialization may be either in loop statement or outside the loop.	Initialization is always outside the loop.
Once the statement(s) is executed then after increment is done.	Increment can be done before or after the execution of the statement(s).
It is normally used when the number of iterations is known.	It is normally used when the number of iterations is unknown.
Condition is a relational expression.	Condition may be expression or non-zero value.
It is used when initialization and increment is simple.	It is used for complex initialization.
For is entry controlled loop.	While is also entry controlled loop.
for (init ; condition ; iteration) { statement(s); }	while (condition) { statement(s); }

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>
int 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
return 0;
} //end main
```

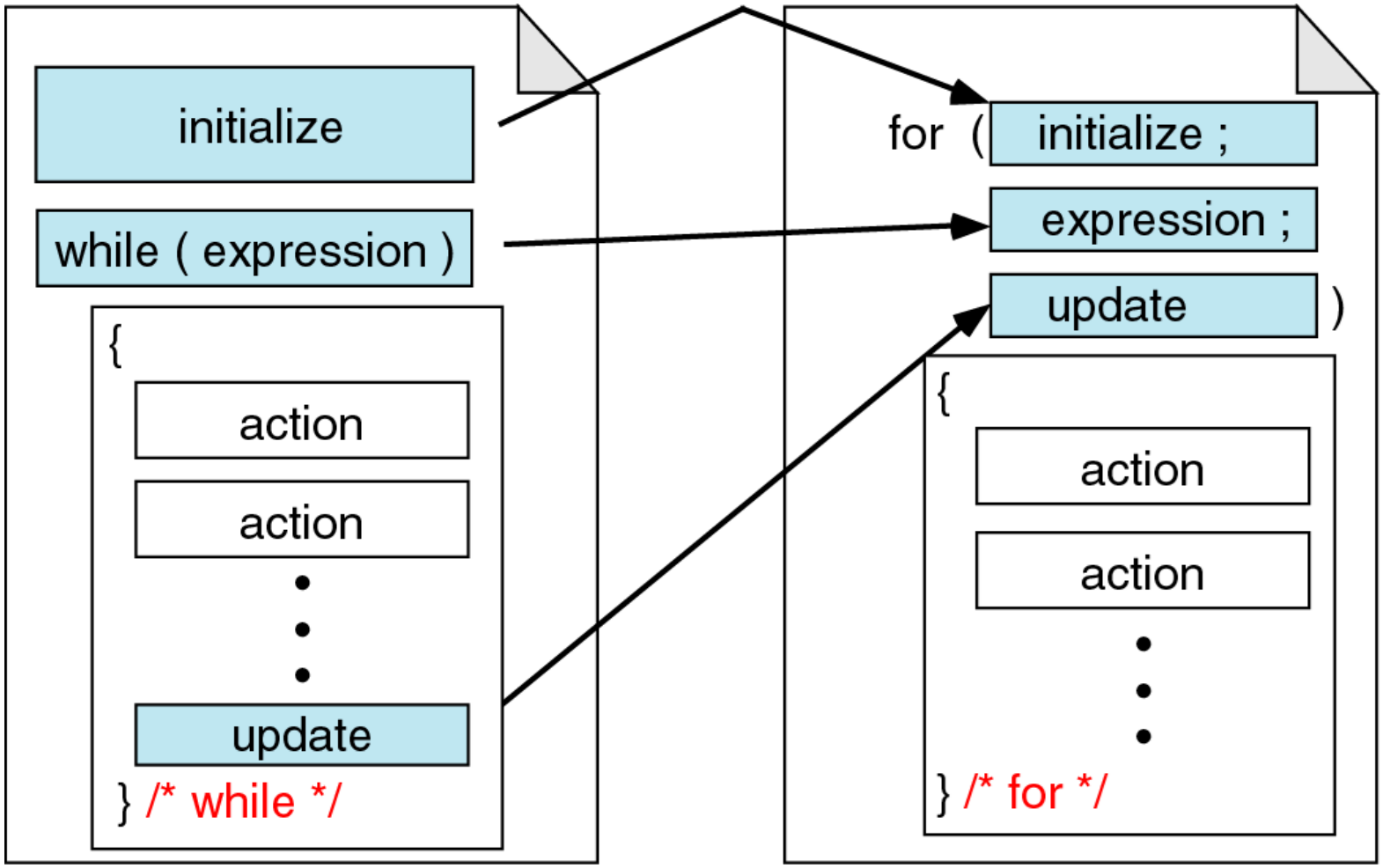
```
Enter a number
4
The tables from 1 to 4
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
```

Program to display a pattern.

```
#include<stdio.h>
int 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");
 }
return 0;
}
```

```
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:

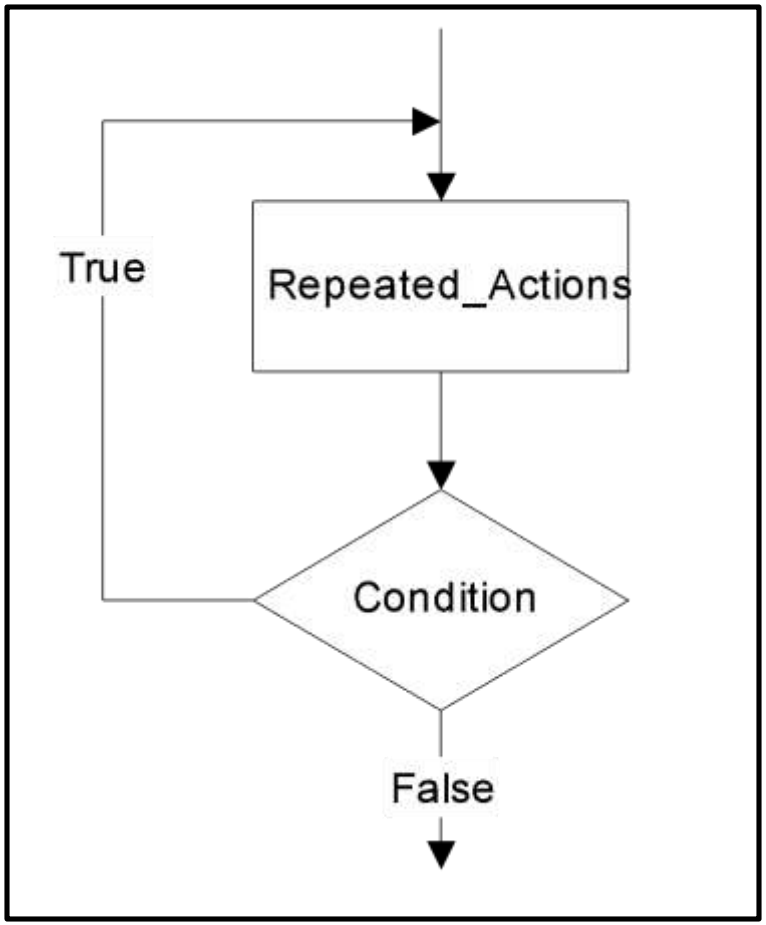
Syntax

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

- The *statement* executed at least one time(even if the condition is false)
- For second time, If the **condition** is true, then the *statement* is repeated else the loop is exited.
- Also known as exit-controlled loop, as loop body executes first and then the condition is checked

do…while statement

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

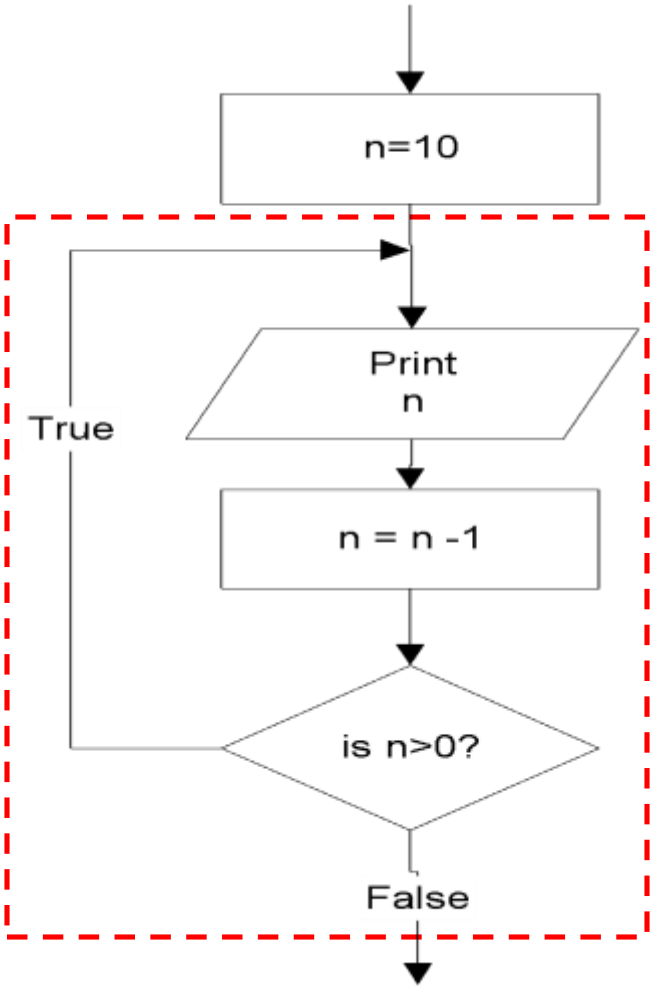


do…while statement

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);**

Q1

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
   do
      printf("In while loop ");
   while (0);
      printf("After loop\n");
   return 0;
}
```

A. In while loop

B. In while loop
   After loop

C. After loop

D. Infinite loop

Q2

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int i = 0;
    do {
        i++;
        printf("In while loop\n");
    } while (i < 3);
   return 0;
}
```

A. In while loop
In while loop
In while loop

B. In while loop
In while loop

C. Nothing will be displayed

D. Compile time error