CONTROL FLOW

Introduction

- **In a program instructions are always executed in the sequential manner.**

- **If programmer wishes to specify the order in which computations are performed then he/ she will need a control flow statement.**

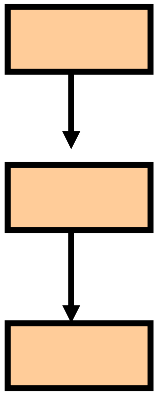
- **Control statements are needed to alter the flow of execution of a program instructions.**

Basic control flow structures

- **Sequential control flow**
- **Selection control flow**
- **Repetitive control flow**

Sequential flow

- **Instructions are executed in serial fashion**
    - To be carried out one after the other...
    - Without hesitation or question



Selection flow

- **An instruction which selects which of the two or more given set of instructions is to be executed, based on a given TRUE/ FALSE condition.**
- **Flow of program is not sequential but depend upon the outcome of given condition whether True/ False.**

Example – car repair

```
if (motor turns)
{
    Check_Fuel
    Check_Spark_Plugs
    Check_Carburettor
}
else
{
    Check_Distributor
    Check_Ignition_Coil
}
```

Should be a true or false condition.

Exercise

Will the following algorithms produce the same output?

Algorithm 1:

```
input Num

if (Num !=0 )

        output 1/Num

else

        output "infinity"
```

Algorithm 2:

```
input Num

if (Num !=  0)
{
                output 1/Num
}

output "infinity"
```

Selection(several conditions)

- **What if several conditions need to be satisfied?**

```
if ( today is Tuesday AND the time is 12.00 noon )

        Go to Programming Lecture Theatre
else

        Go to Annapurna for lunch
```

logical AND (&&)

Solution 1

Solution 2

```
if ( today is Wednesday )
 {
     if ( The time is 12.00 Noon )
      {
        Go to Programming Lecture Theatre
      }
 }
 else
   Go to Annapurana for lunch
```

This is called as NESTED LOOPS

Solution 2

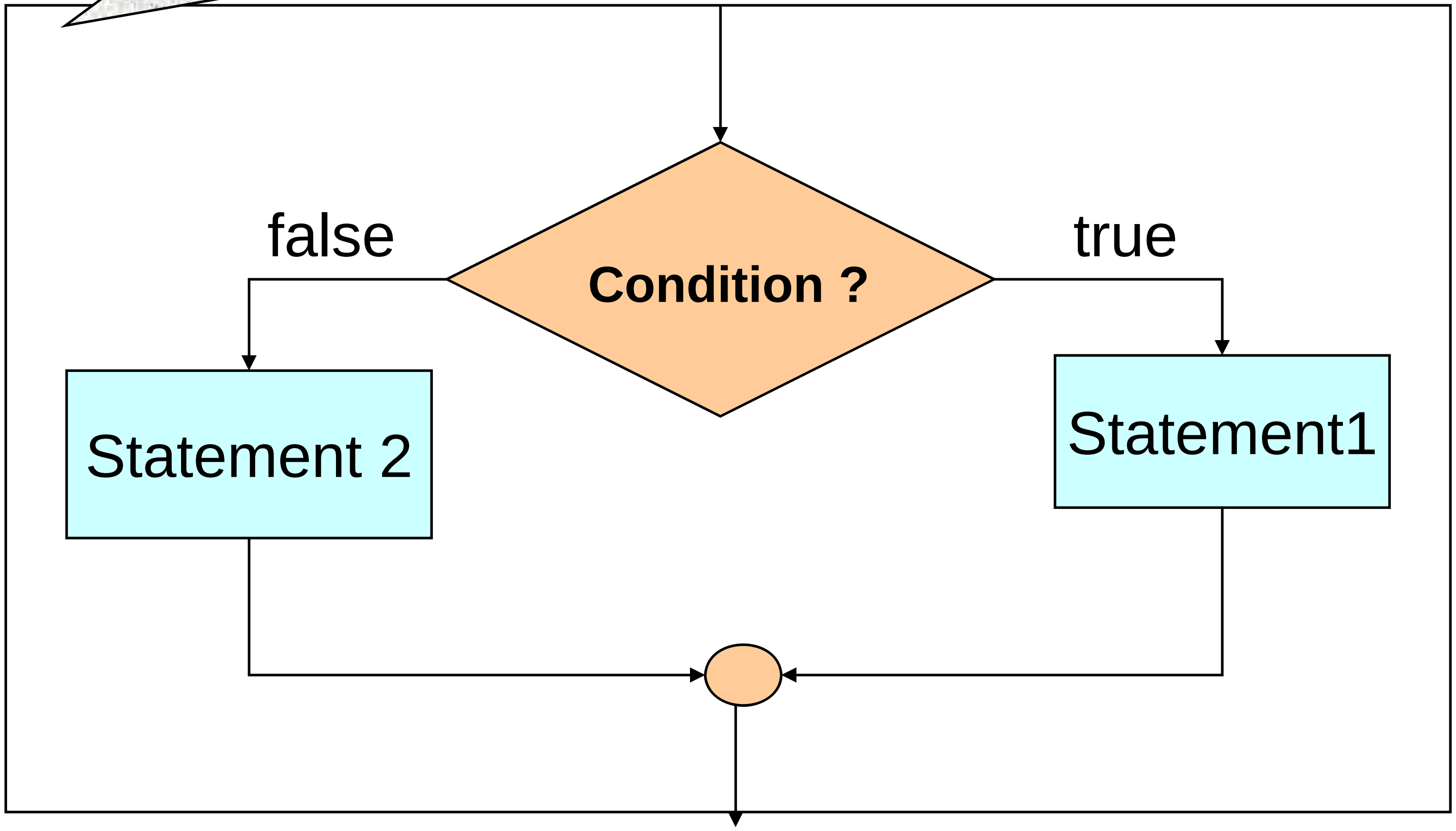
Selection: At least one of the several conditions

What if at least one of several conditions needs to be satisfied?

```
if ( We feel hungry or the time is 12.00 Noon  or the time
is 1 P.M. )
{
            Go to the Annapurana for lunch

 }
```

logical OR (||)

If the condition is true only statement1 will be executed, if the condition is false only statement2 will be executed.Statement1 and statement2 will not be both executed at the same time



The if else statements

- The if-else statement is used to carry out logical test.
- The general form is

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

- If the expression has a non zero(True) value then statement 1 will be executed , otherwise statement 2 will be executed.
- The *else part* of the *if statement* is optional.

- The value of the expression is 1, if the it is true.
- If the expression is false, the value returned is 0.
- Always remember
    - True : Non zero value (usually 1)
    - False: Zero value(0)

Single statement

```
If   ( a > b )
   printf (“ a is greatest”);
else
   printf (“ b is greatest);
```

Compound statement

```
if   (a < b)
 {
        t = a;
        a = b;
        b = t;
 }
else
  {
     printf(“ Wrong choice”);
      printf(“Re-enter the data”);
  }
```

x	10
y	10
z	20

```
#include <stdio.h>
int main()
{

        int x, y,z ;
        scanf("%d %d", &x, &y);
      if ( x = = y)
        z =  x + y ;
        else
        z = x % y;

return 0;
}
```

It d

A

The condition (x = = y) is true, 10 is equal to 10, the statement
z = x + y;
will be executed.

The value 20 is stored to memory z. The program leave the if/else statement and then move to next statement.

The program ends here.

else: optional

- else part of If statement is optional but often useful.

- Example:

```
if (a < b)
        x = a;
else
         x = b;
```

Problem

Problem: Print the tax% for a given income based on the following table :

income	tax in %
< 1,00,000	0%
1,00,000 <=income< 1,50,000	10%
1,50,000<=income< 2,50,000	20%
income>=2,50,000	30%

Direct Solution

```
if ( income<100000 )
    printf( “No tax.” );

if ( income >= 100000 && income < 150000 )
    printf(“10%% tax.”);

if ( income >= 150000 && income < 250000 )
     printf(“20%% tax.”);

if ( income >= 250000)
    printf(“30%% tax.”);
```

Exercise

- **Write the if else version of the above problem.**

Problem: Print the net income based on the tax% given in the previous slides.

Practice Question

- Write a C program to take three numbers as input from user and Find the largest of three numbers.

Common Errors

- Omission of & in scanf statement.
- Omission of Format specifier in the printf statements.
- Using & in printf statement.
- Adding ; (semi colon) after expression in if statement.
- Incorrect use of curly braces in if statement.
- Omission of parenthesis in expression of if statement.
- Usage of format specifiers in printf without identifier/variable name.

- Using assignment operator in place of equality.

```
if( a =10)
      printf(“value of a is equal to 10”);
  else printf(“value of a is not equal to 10”);
```

- In above example irrespective of the value of a **first printf** will always be executed.

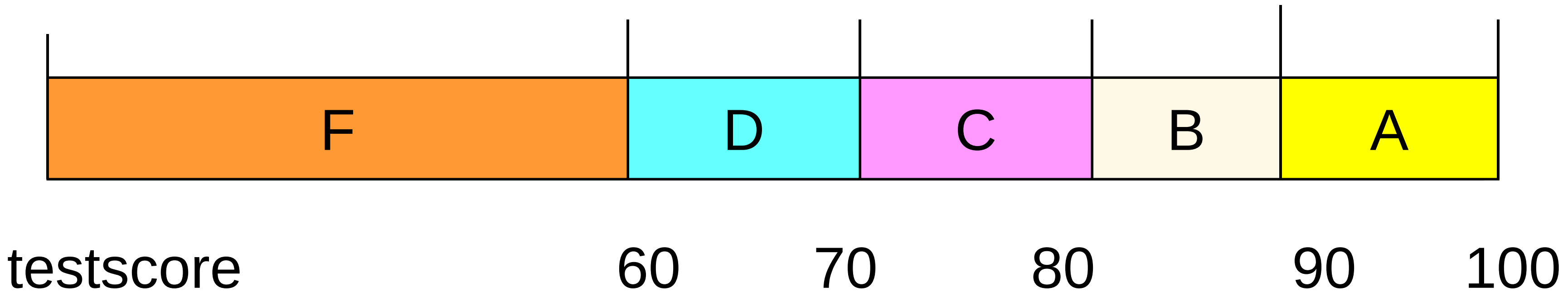
- Error is while **comparison** one should use **==** equality operator.

Exercise

Problem: Print the net income based on the tax% given in the previous slides.

tion

```
if (testscore >= 90) printf(“ Your grade is A”);
    else if (testscore >= 80) printf(“Your grade is B”);
           else if (testscore >= 70) printf”(Your grade is C”);
                  else if(testscore >= 60) printf(“Your grade is D”);
                           else printf(“Your grade is F”);
```



Nested if/else statement walk through

testscore	63

```
#include <stdio.h>
int main()
{
 int testscore;
 scanf("%d", &testscore);
 if (testscore >= 90) printf(" Your grade
     else if (testscore >= 80) printf(
         else if (testscore >= 70) printf"(Your grade is C
               else if(testscore >= 60) printf("Your grade is D");
                      else printf("Your grade is F");
 return 0;
}
```

D

The condit

printf p

is not exec

Your grade is D

The condtion (63>= 70)is false, the statement printf("Your grade is C"); will be not executed

,

be executed

will not be executed

The programs ends here

Conditional operator (? :)

- It provides an alternative way to implement an **if else** statement.

```
If ( a > b )
        z =a;
else
        z=b;
```

- This is a ternary operator with the general syntax of :

```
exp1 ?  exp2   :   exp3;
```

(? :) operator

- expr1 ? expr2 : expr3

  In above expr1 is evaluated first. If it is non zero (true), then the expression expr2 is evaluated otherwise expr3 is evaluated.

$$\begin{tikzcd}[column sep=large] \text{expr1} \arrow[rr, bend left, "\text{true}"] \arrow[rrrr, bend right, "\text{false}"'] & ? & \text{expr2} & : & \text{expr3}\end{tikzcd}$$

Example:

```
z = ( a > b ) ? a : b ;   /* z = max( a, b) */
```

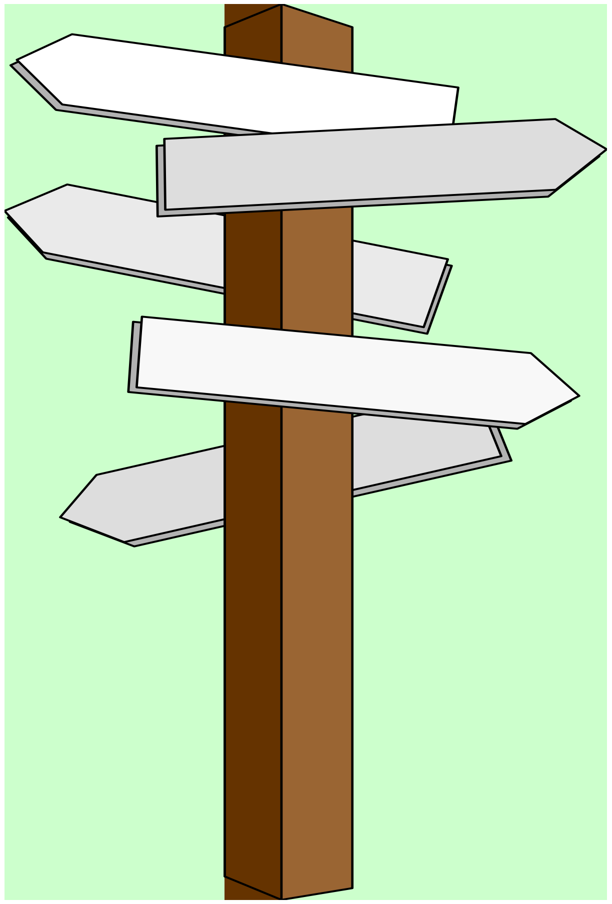
Example

- **Assume x=8,y=2.0,I=5,j=7sum=0,a=1;**
- **Y = ((x>=10) ? 0 : 10) ;**

- **Res = ((I < j) ? sum +I : sum + j) ;**

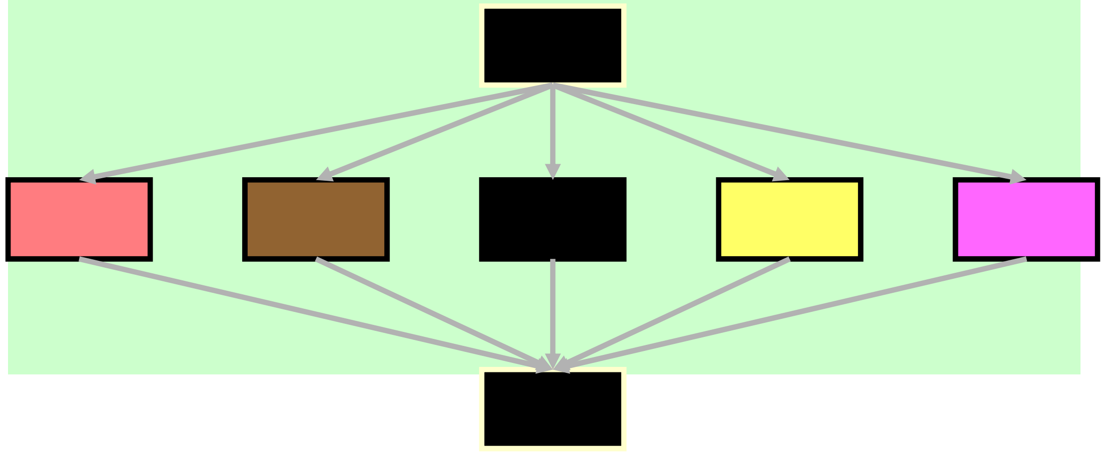
- **Q = ((a = = 0) ? 0 : (x/y)) ;**

The *switch* Statement



The *switch* statement

- causes one group of statements to be chosen from several groups
- based on the value of an expression



Switch

- A switch is a form of conditional statement.

- It is specifically designed to be useful in multi-way choice situations.

- Instead of a condition, there is a value which is tested, and a series of **cases** of which only one may be chosen.

The *switch* Statement

The general form is

switch (expression)

{

case *e1*: __________

case *e2*: __________

. . . .

case *em*:__________

}

Can result in an integer value or char value.

Cases

- A case is a section of code within the switch statement. A case is executed only if the switch expression has a specified value

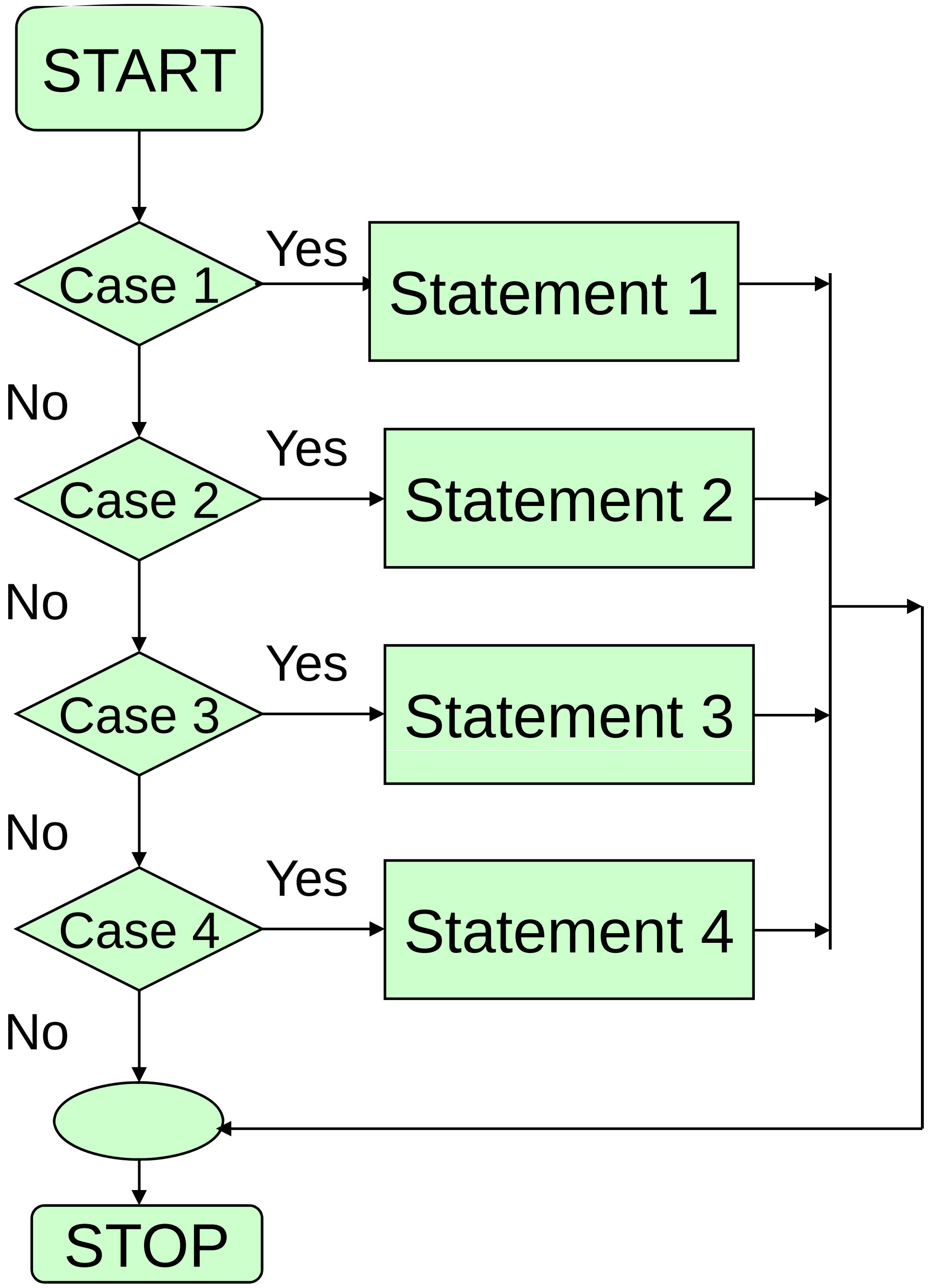
```
case value:
    /* a sequence of statements*/
```

Example

```
main()
{
    int i;
     scanf (“%d”, &i);
    switch (i)
    {
          case 1: printf(“I am in case 1 ”);
          case 2: printf(“I am in case 2 ”);
          case 3: printf(“I am in case 3 ”);
          default: printf(“I am in default ”);
    }
}
```

Let i = 2   The output will be

I am in case 2 I am in case 3 I am in default.



```
switch (choice)
{
        case 1:
                statement 1;
                break;
        case 2:
                statement 2;
                break;
        case 3:
                statement 3;
                break;
        case 4:
                statement 4;
                break;
}
```

Break Statement

- The *break* statement causes a transfer of control out of the entire switch statement, to the first statement following the switch statement.

- The *break* statement is written as

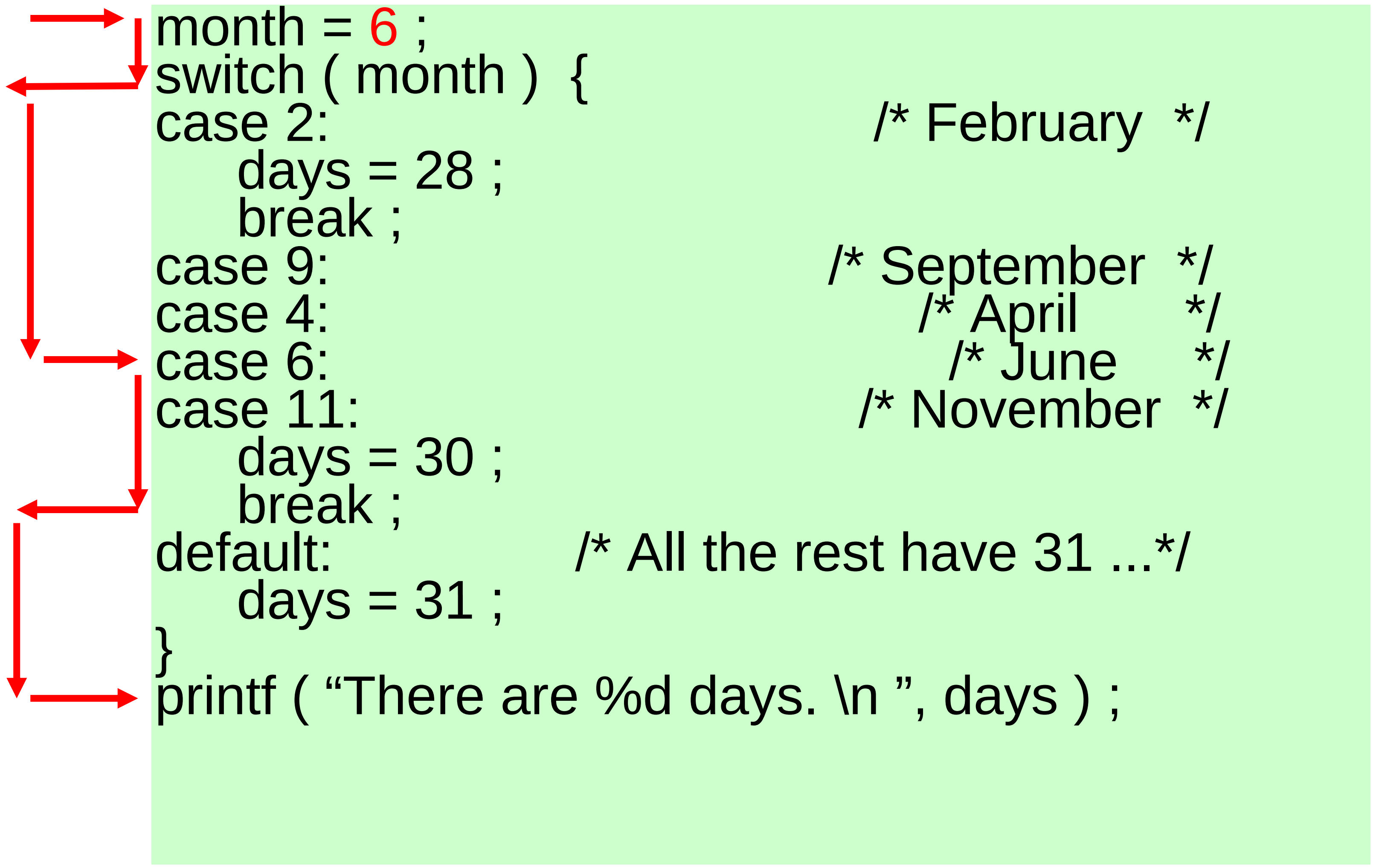
```
break;
```

```
main()
{
   int i = 2;
   switch (i)
   {
        case 1: printf(“I am in case 1 ”);
                break;
        case 2: printf(“I am in case 2 ”);
                break;
        case 3: printf(“I am in case 3 ”);
                break;
        default: printf(“I am in default ”);
   }
}
```

The output will be

I am in case 2

switch: Flow of Control



```
month = 6 ;
switch ( month )  {
case 2:                                          /* February  */
     days = 28 ;
     break ;
case 9:                                       /* September  */
case 4:                                            /* April        */
case 6:                                              /* June     */
case 11:                                        /* November  */
     days = 30 ;
     break ;
default:                   /* All the rest have 31 ...*/
     days = 31 ;
}
printf ( “There are %d days. \n ”, days ) ;
```

```
/* The program assigns a value to y related to x, depending
   upon the value of flag*/
main( )
{   int flag;
    float y, x;
    scanf("%d  %f", &flag, &x);
    switch (flag)
          { case -1:      y = x * x;
                          break;
            case 0:       y = abs(x);
                          break;
            case 1:       y = x;
                          break;
            case 2:       y = 2 * (x-1);
                          break;
            case 3:       y = 2 * (x-1);
                          break;
            default:      y = 0;}
    printf("%f", y);    }
```

Exercise

- Write one program using the switch statement to compute areas of any of the following:
  - Area of a circle
  - Area of a rectangle
  - Area of a square
  - Area of a triangle

Approach

↖ decide the input variables and their type

↖ decide the value of expression and its type

```
main( )
{   float len, width, area,rad;
    int choice;
    scanf("%d", &choice);
    switch (choice)
    {
        case 1: scanf("%f", &rad);
                area = 3.14 * rad * rad;
                printf("Area of circle = %f", area);
                break;
        case 2: scanf("%f %f", &len, &width);
                area = len * width;
                printf("Area of rectangle = %f", area);
                break;
```

Document

1 for circle

2 for rectangle

3 for square

4 for triangle

Good idea to prompt for a value of choice between 1 and 4

Still you must check for a valid value.

```
case 3:  scanf("%f", &len);
            area =   len * len;
            printf("Area of square = %f", area);
            break;
case 4:
            scanf("%f %f", &len, &width);
            area = 0.5 * len * width;
            printf("Area of triangle = %f", area);
            break;
default:printf("The choice should be between 1 to 4");
         }
}
```

More on switch

- The cases in switch could be in any order.

```
switch(i)
{
	case 121: ________
	case 7: __________
	case 22: _________
	default: _________
}
```

More on switch

- Can use *char* value in *case* and *switch*.
- e.g.

```
main( )
{
      char c = 'x';
      switch(c)
            {
                  case 'v': ________
                  case 'a': ________
                  case 'x': ________
                  default : ________
            }
}
```

More on switch

- Can mix *integer* and *character* constants in different *cases* of a *switch*.
- e.g.

```
main( )
{      int c = 3;
switch(c)
{
       case 1: ________
       case ‘a’: ________
       case 3: ________
       default : ________
}
```

Advantage of *switch* over *if*

- more structured program
- manageable indentations
- better way of writing programs

Disadvantage of *switch* over *if*

- An expression resulting in a float value is not allowed in switch.

- Cannot have expressions in cases. For example, one cannot have a switch which looks like

```
case (i<= 20):
```

Looping

for (……..)

do while (cond …….)

while (cond………)

What is a Looping?

The term "looping" describes the way in which the program executes statements over and over again, before exiting the loop and continuing with program flow.

Loops in C

In C, there are three types of loops: for loops, while loops and do while loops.

It is possible to simulate each type by writing code using other loops.

Repetition

Repetition : 3 types of loops

-
  ```
  while (<cond>)
  <statement>;
  ```

-
  ```
  do
        <statement>;
  while (<cond>);
  ```

-
  ```
  for (<init>; <cond>; <iterate>)
        <statement>;
  ```

While loop

- The general syntax of while loop is:

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

- Loop is executed only when the condition given is TRUE that is non zero.

- When condition is false control passes to the line of code immediately following the loop

Example

```
int count=0;
while(count < 10 )
{
printf(“%d”,count);
count ++;
}
```

- It checks the test condition at the top of loop.
- If condition is true then only loop will be executed otherwise control will come out of the loop.

Example -- Count to 10

Print out numbers 0 to 9

```
set count to 0
while ( count is less than 10 )
{
    output count
    add 1 to count
}
```

```
int main()
{

   return 0;
}
```

Example -- Count to 10 (cont)

Print out numbers 0 to 9

```
set count to 0
while ( count is less than 10 )
{
    output count
    add 1 to count
}
```

```c
#include <stdio.h>


int main()
{








  return 0;
}
```

Example -- Count to 10 (cont)

Print out numbers 0 to 9

```
set count to 0
while ( count is less than 10 )
{
    output count
    add 1 to count
}
```

```c
#include <stdio.h>

/* Print out numbers 0 to 9 */

int main()
{






  return 0;
}
```

Example -- Count to 10 (cont)

Print out numbers 0 to 9

```
set count to 0
while ( count is less than 10 )
{
    output count
    add 1 to count
}
```

```
#include <stdio.h>

/* Print out numbers 0 to 9 */
int main()
{
      int count;

      count = 0;
      while ( count < 10 )
      {
            printf(“%d\n”,
count);
            count=count+1;
      }
  return 0;
}
```

Example -- Count to 10 (cont)

Print out numbers 0 to 9

```
set count to 0
while ( count is less than 10 )
{
    output count
    add 1 to count
}
```

```
#include <stdio.h>

/* Print out numbers 0 to 9 */
int main()
{
      int coun

      count = 0;
      while ( count < 10 )
      {
            printf(“%d\n”,
count);
            count=count+1;
      }
  return 0;
}
```

Assignment of a value (right expression) to a variable (left).

Example -- Count to 10 (cont)

Print out numbers 0 to 9

```
set count to 0
while ( count is less than 10 )
{
    output count
    add 1 to count
}
```

```
#include <stdio.h>

/* Print out numbers 0 to 9 */
int main()
{
      int count;

      count = 0;
      while ( count < 10 )
      {
      printf(“%d\n”, count);
      count=count+1;
      }
  return 0;
}
```

No semi-colon here!

Example -- Count to 10 (cont)

Print out numbers 0 to 9

```
set count to 0
while ( count is less than 10 )
{
    output count
    add 1 to count
}
```

```c
#include <stdio.h>

/* Print out numbers 0 to 9 */
int main()
{
      int count;

      count = 0;
      while ( count < 10 )
      {
      printf("%d\n", count);
            count=count+1;
      }
  return 0;
}
```

Example -- Count to 10 (cont)

Print out numbers 0 to 9

```
set count to 0
while ( count is less than 10 )
{
    output count
    add 1 to count
}
```

```
#include <stdio.h>

/* Print out numbers 0 to 9 */
int main()
{
      int count;

      count = 0;
      while ( count < 10 )
      {
        printf("%d\n", count);
            count=count+1;
      }
  return 0;
}
```

Format string

Example -- Count to 10 (cont)

Print out numbers 0 to 9

```
set count to 0
while ( count is less than 10 )
{
    output count
    add 1 to count
}
```

```
#include <stdio.h>

/* Print out numbers 0 to 9 */
int main()
{
      int count;

      count = 0;
      while ( count < 10 )
      {
      printf(“%d\n”, count);
            count=count+1;
      }
  return 0;
}
```

WHILE Loop

```
#include <stdio.h>

int main() {
 int i=10; /* initialize variables */
 int j=0;  /* part a of a for loop */

 while (i!=j) { /* test for condition
                   part b of for loop */
    printf("%d - %d = %d\n", i, j, i-j);
    i--;  /* do something to variables */
    j++;  /* part c of for loop        */
 }
 return 0;
}
```

Example

```
while (  ( ch =getchar( ) ) ! = ‘A’ ) ;
```

```
char ch;
ch =‘Z’;
while( ch !=‘A’)
ch = getchar( );
```

Will this works?

```
Count =10;
while (Count)
 {
   printf (“The value of Count is : %d”, count);
   count ++;
  }
```

- This is an infinite loop.

Will this Work?

```
Count =5;
while (Count)
  {
          printf (“The value of Count is : %d”, count);
          count--;
   }
```

Will this Work?

```
count = 10;
counter = 0;
while (count)
  {
        printf (“The value of Count is : %d”, count);
        count++;
        counter++;
        if (counter > 10) break;
   }
```

Do While loop

The general format is

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

- Test condition is checked at the bottom of the loop, it means the loop executes at least once.

- The most common use of this loop is in menu selection function.

Example

```
#include<stdio.h>
/ * this loop will be executed only once */
int main( )
  {
      int count=0;
   do
      {
       printf ("The value of Count is : %d\n", count);
       count -  - ;
      }
   while (count>0);

return 0;
  }
```

do-while Loop

```
main)()
{
/* Count from 1 to n */
 int n,i = 1;
 scanf (“%d”,&n);
 do
{
  printf("%d ", i++);
} while (i <= n);
}
```

Exercise

- For a class of 30 students, you are required to write a program that should accept the marks of one student in five subjects at a time, calculates his average marks, and displays the the grade using given table. Then it should ask the marks for second student and repeats the same task for remaining students of the class.

Percentage	**Grade**
> =80	**A**
>=75	**B**
>=55	**C**
<54	**D**

Home Assignments

Greatest of n numbers

Odd number generation up to 50 other than multiples of 3 and 7

Factorial of a number

Generate the series A, AA, AAA, AAAA….n

Generate the series
AA,AAAA,AAAAAA….

```
main ( )
{  int n, number,greatest = 0;
  int count = 1;
  printf(“Enter the total number's you want to test”);
  scanf(“%d”,&n);
  while (count <= n)
   {printf (“ enter the new number : ”);
     scanf (“%d”, &number);
      if (greatest < number)
           greatest = number;
    count++;     }
 printf (“ the greatest number is  : %d” , greatest);
}
```

The FOR loop

- The general form of this FOR statement is

  for(*initialization ; condition ; increment*)

        statements ;

- Initialization : Used to set the loop control variable(s).
- Condition : It is a relational expression that determines when the loop exits.The loop is repeated as long as condition is true.
- Iteration(Increment/decrement) : It determines how the value of control variables changes each time the loop is repeated.

  ✓ All the three conditions are separated by semicolon(;)

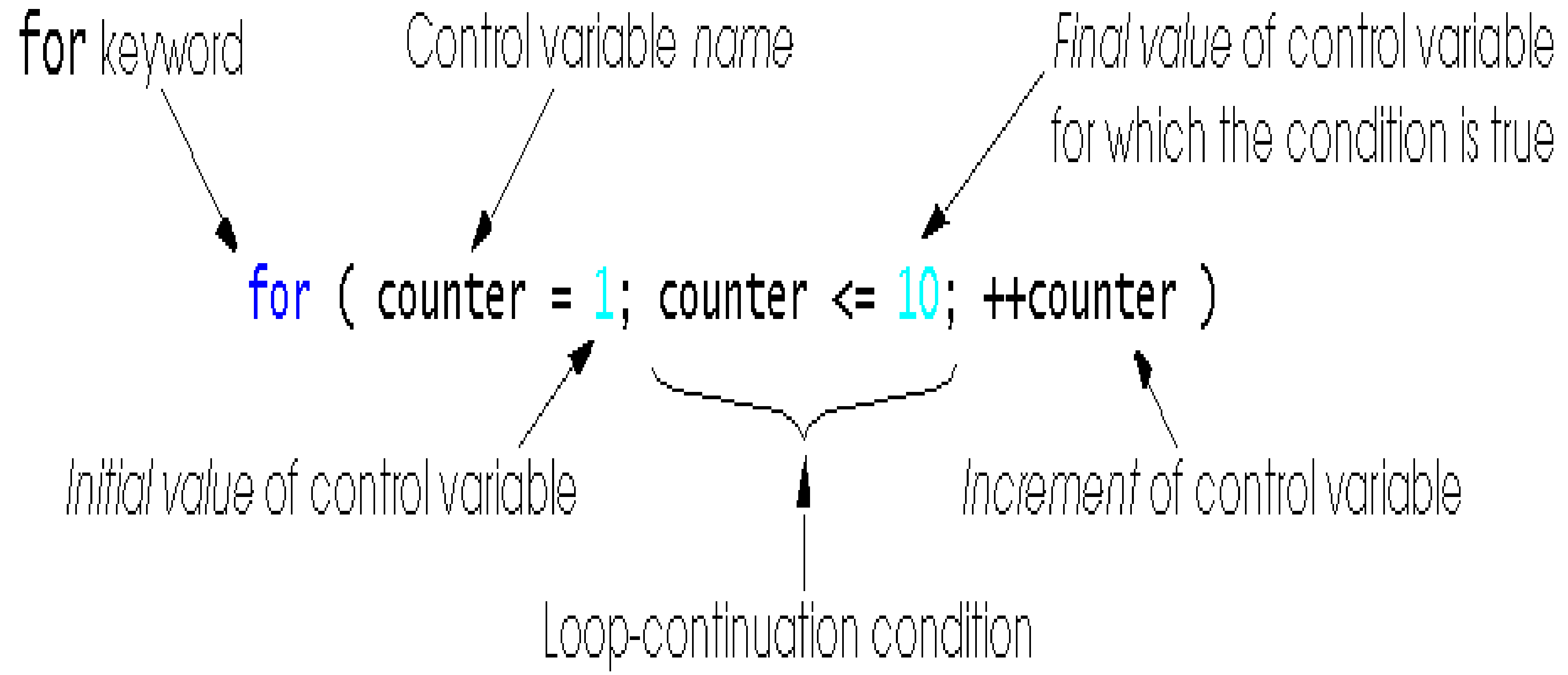
“for” Loop

Syntax:

```
for (a ; b ; c)
{
 /* statements */
}
```

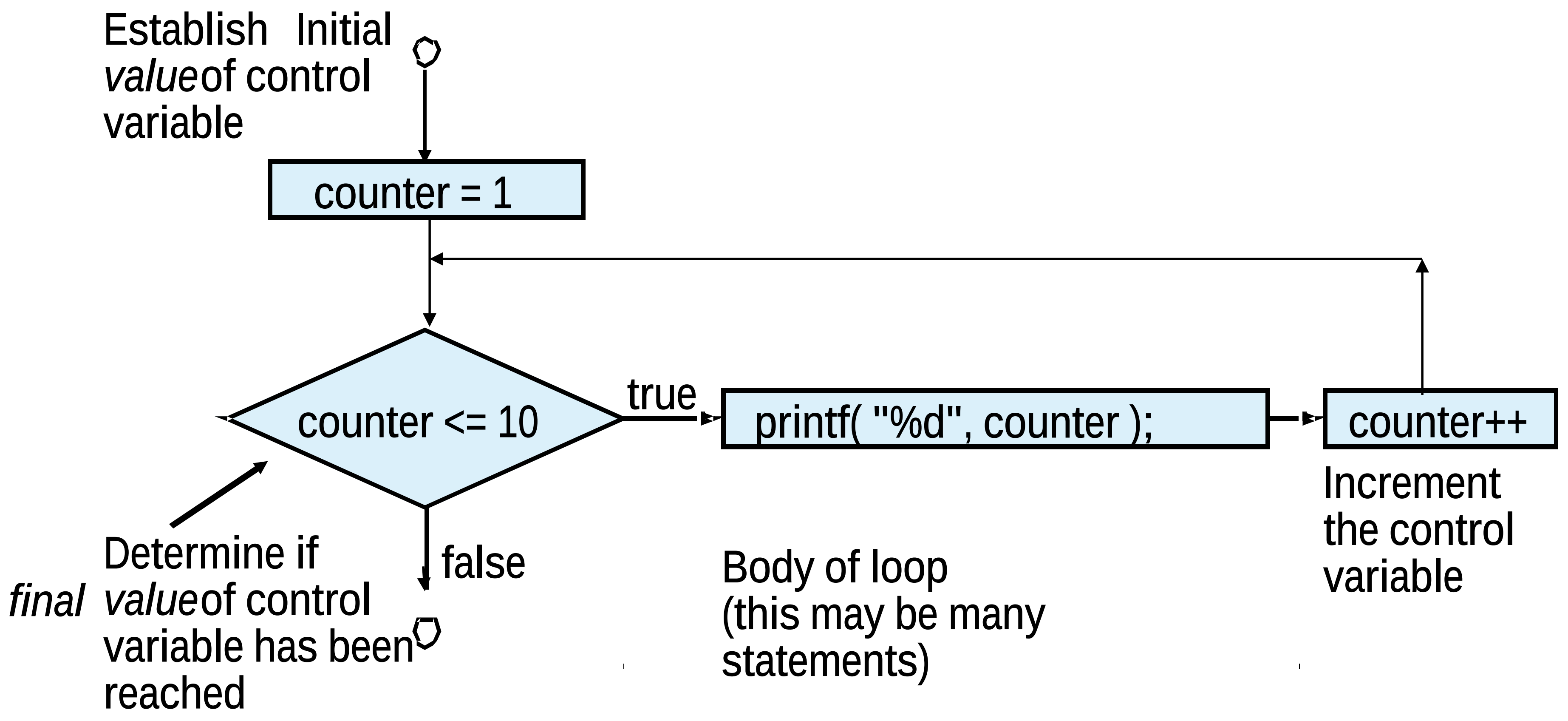
- **a(**Initialization **),** **b(**Condition) **and** **c(**Iteration(Increment/decrement) **are expressions that are evaluated at different times.**

Example



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

Flow of execution



Keep in mind

- Initialization/ increment expression is optional however do not forget to put (;) example

```
for (   ;  condition   ;    )
```

- for statement can have more than one initialization or increment condition. e.g.

```
for (i=10, j=0   ;  condition   ;    )
for (i=10, j=0   ;  condition   ; i--, j++   )
```

- Test condition is must inside for.You can not omit it.

Example 1

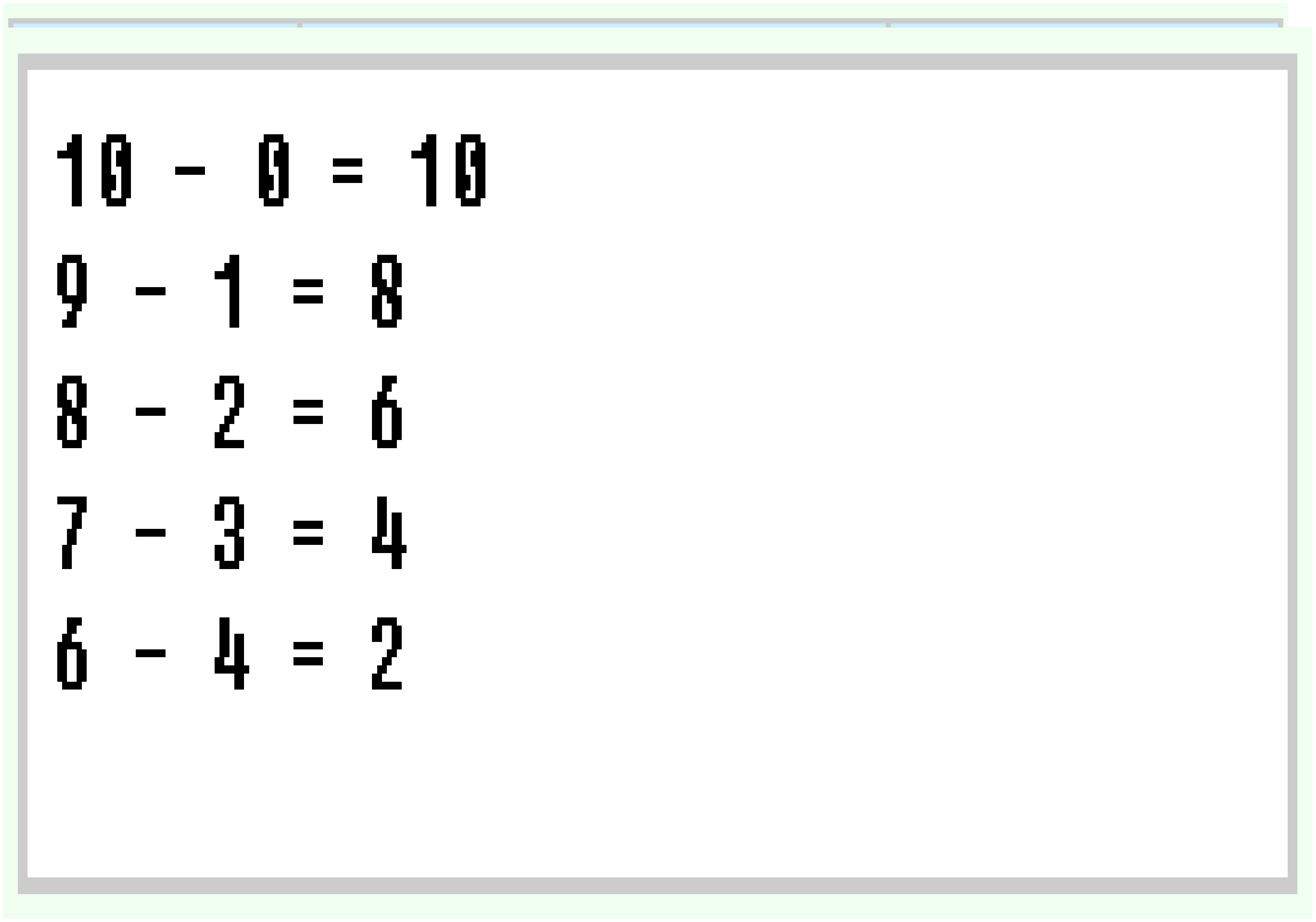
```
int x ;
for ( x=1 ; x<=100 ; x++ )
     printf(“%d”, x);
```

```
int   x,  z ;
for ( x=100 ; x!=65 ; x-=5 )
 {
     z = x*x ;
    printf(“the square of %d  is , %d”, x, z );
  }
```

Example 2

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

Example 2 Cont'd



```
10 - 0 = 10
9 - 1 = 8
8 - 2 = 6
7 - 3 = 4
6 - 4 = 2
```

Example 3

FIND THE AVERAGE OF MARKS OF 6 SUBJECTS

```
 int count,mark;
 float  average, total =0;
 for ( count = 0 ; count <6; count++ )
     {
          printf (“\n Enter the mark : ”);
          scanf (“%d”, &mark);
          total = total + mark;
     }
printf (“ the total mark is : %f”,total);
average = total / (count);
printf (“the average mark is : %f”,average);
```

For statement: observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.

    If x = 2 and y = 10

    ```
    for ( j = x  ; j <= 4 * x * y  ; j += y / x )
    ```

    is equivalent to

    ```
    for ( j = 2 ; j <= 80  ; j += 5 )
    ```

Infinite loop

```
for(  ;   ;  )
   printf(“ what will be the output”);
```

- When conditional expression is absent, it is assumed to be TRUE
- Even if initialization and increment expressions are present it will be an infinite loop.

Class Exercise

- Write a program to find Greatest of n numbers using for loop.

Greatest of n numbers

```
int num,largest,tot_num,count;
printf(“ Enter the total numbers”);
scanf(“%d”,tot_num);
printf(“ Enter the first number”);
scanf(“%d”,num);
largest=num;
        for ( count = 1 ; count <tot_num; count++ )
        {

        }
    printf (“ the largest number is : %d”,largest);
```

Continue statement

- Used to bypass the remainder of the current pass through a loop

- Loop does not terminate when a continue statement is encountered. Rather the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop

- Can be included within a while, a do-while or a for statement

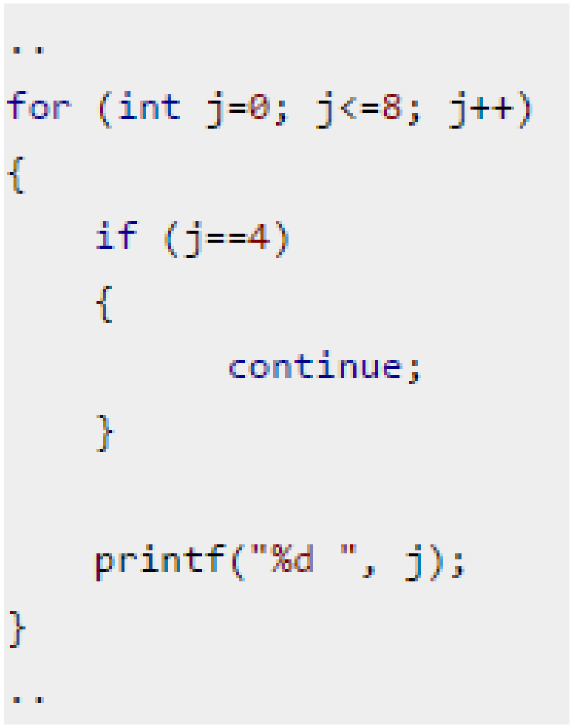
- It’s syntax is written as:

continue; /*without any embedded statements

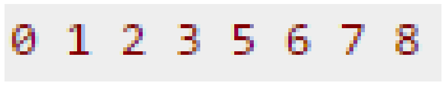
or expressions*/

Example:

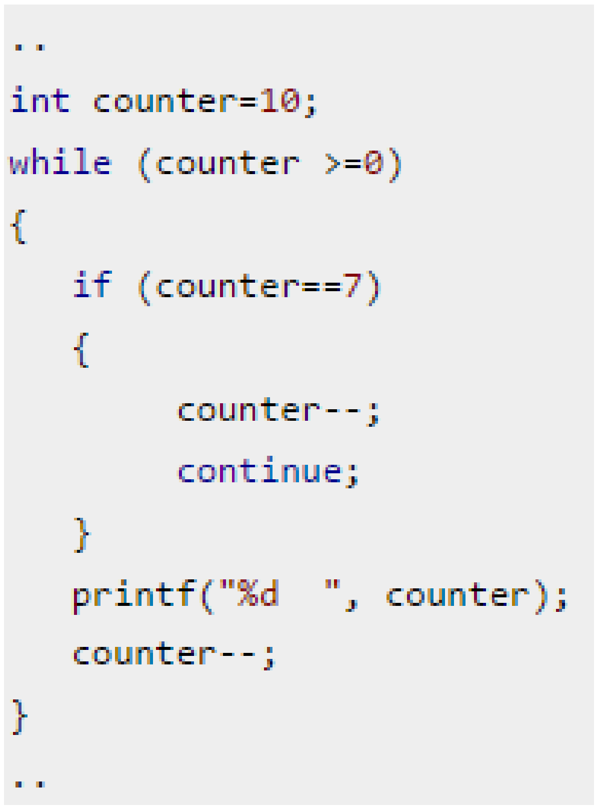
```
#include<stdio.h>
main()
{
int n, count, navg=0;   float x,average,sum=0;
printf(“how many no”);
scanf(“%d”,&n);
for(count=1;count<=n;++count) {
printf(“x= “);
scanf(“%f”,&x);
if(x<0)
continue;
sum+= x;
++navg;}
average=sum/navg;
printf(“\n the average is %f\n”,average);
}
```



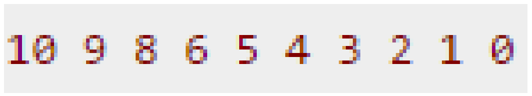
```c
..
for (int j=0; j<=8; j++)
{
    if (j==4)
    {
          continue;
    }

    printf("%d ", j);
}
..
```



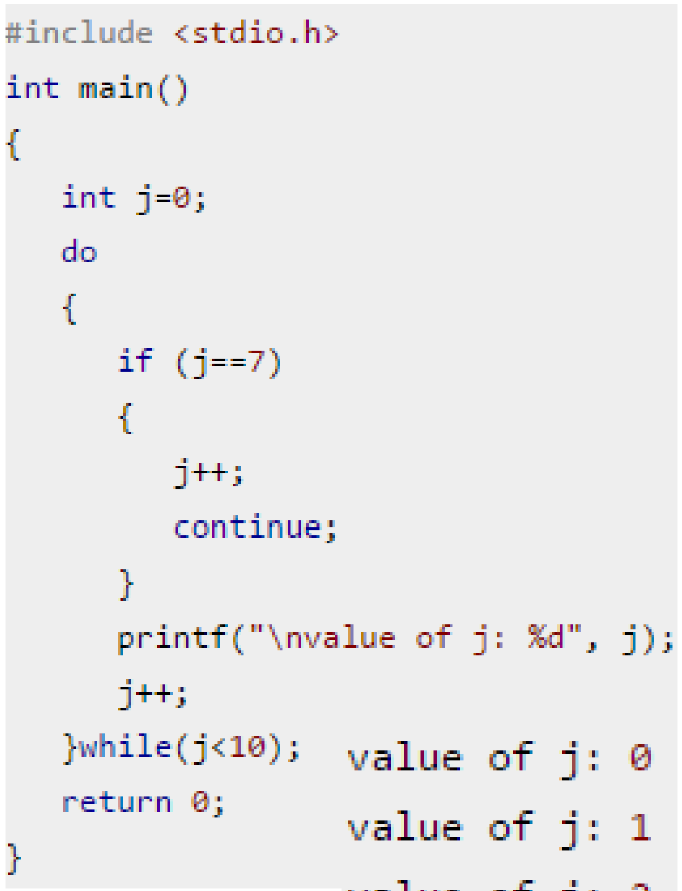
```
0 1 2 3 5 6 7 8
```



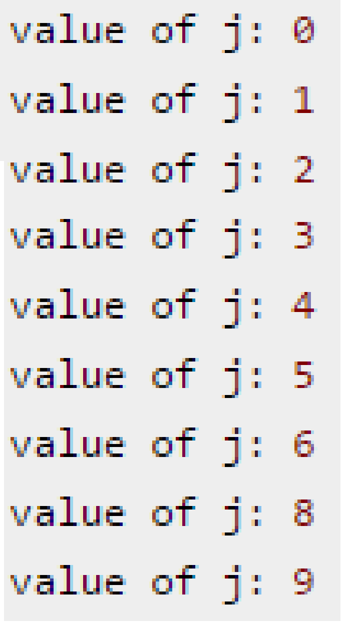
```c
..
int counter=10;
while (counter >=0)
{
   if (counter==7)
   {
        counter--;
        continue;
   }
   printf("%d  ", counter);
   counter--;
}
..
```



```
10 9 8 6 5 4 3 2 1 0
```



```c
#include <stdio.h>
int main()
{
   int j=0;
   do
   {
      if (j==7)
      {
         j++;
         continue;
      }
      printf("\nvalue of j: %d", j);
      j++;
   }while(j<10);
   return 0;
}
```



```
value of j: 0
value of j: 1
value of j: 2
value of j: 3
value of j: 4
value of j: 5
value of j: 6
value of j: 8
value of j: 9
```

The goto Statement

- The goto statement is used to alter the normal sequence of program execution by transferring control to some other part of the program.
- The goto statement is written as

  goto label;

  where label is an identifier that is used to label the target statement to which control will be transferred.
- The target statement is written as:

  label: statement

Example

```
main( )
{  int goals;
    printf("Enter the number of goals scored against India");
    scanf("%d",&goals);
    if (goals <=5)  goals = 3
          goto sos;
    else
          goto good;  goals = 7
    sos: printf("To err is human");
    exit( ); /*terminates program execution*/
    good: printf("About time hockey players learnt C and said
                     goodbye to hockey");
}
```

The exit( ) Function

- The exit() function is a standard library function
- This function terminates the execution of the program
- The exit() function requires the header file stdlib.h.

More about *goto*

Most common application are

- Branching around statement or group of statements under certain condition
- Jumping to the end of a loop under certain conditions, bypassing the remainder of the loop during the current pass.
- Jumping completely out of a loop under certain conditions, thus terminating the execution of a loop.

More about *goto*

- No two statements in a program can have the same label.
- Any number of goto’s can take the control to the same label.

Avoid goto's…

- A goto statement makes a program
  - unreliable
  - unreadable
  - hard to debug

- Goto tends to encourage logic that skips all over the program whereas the structured features in C require that the entire program be written in an orderly, sequential manner.

Home Assignments

1. Write a C program for printing the multiplication table of a number using for loop.

Comparison of three Loops

for

```
for (n=1;n<=10;++n)
{
-----------
-----------
}
```

while

```
n=1;
while(n<=10)
{
-----------
-----------
n=n+1;
}
```

do while

```
n=1;
do
{
-----------
-----------
n=n+1;
}
while (n<=10);
```

Auto Type conversions

C can sometimes convert between different types for you

- smaller to larger
  - char to int
  - int to long
- less precise to more precise
  - float to double
  - int to double

Auto Type conversions….

-Final result of expression is converted to variable type on left hand side of assignment operator.However some changes occur if:

- float to int-truncation
- double to float-rounding off

How do you do type conversions?

By using type conversions.

Cast operator can be used for converting the type of an expression.

Syn:

(type) variable name

Example

```
int aa = 28, bb = 22,cc;
float xx = 10.55, yy = 20.50,zz;
cc = (int) xx + (int) yy;
zz = (float) bb / aa;
printf ( “%d”, cc);
printf (“%f”, zz);
```