UES103

Programming for Problem Solving

Saif Nalband, PhD

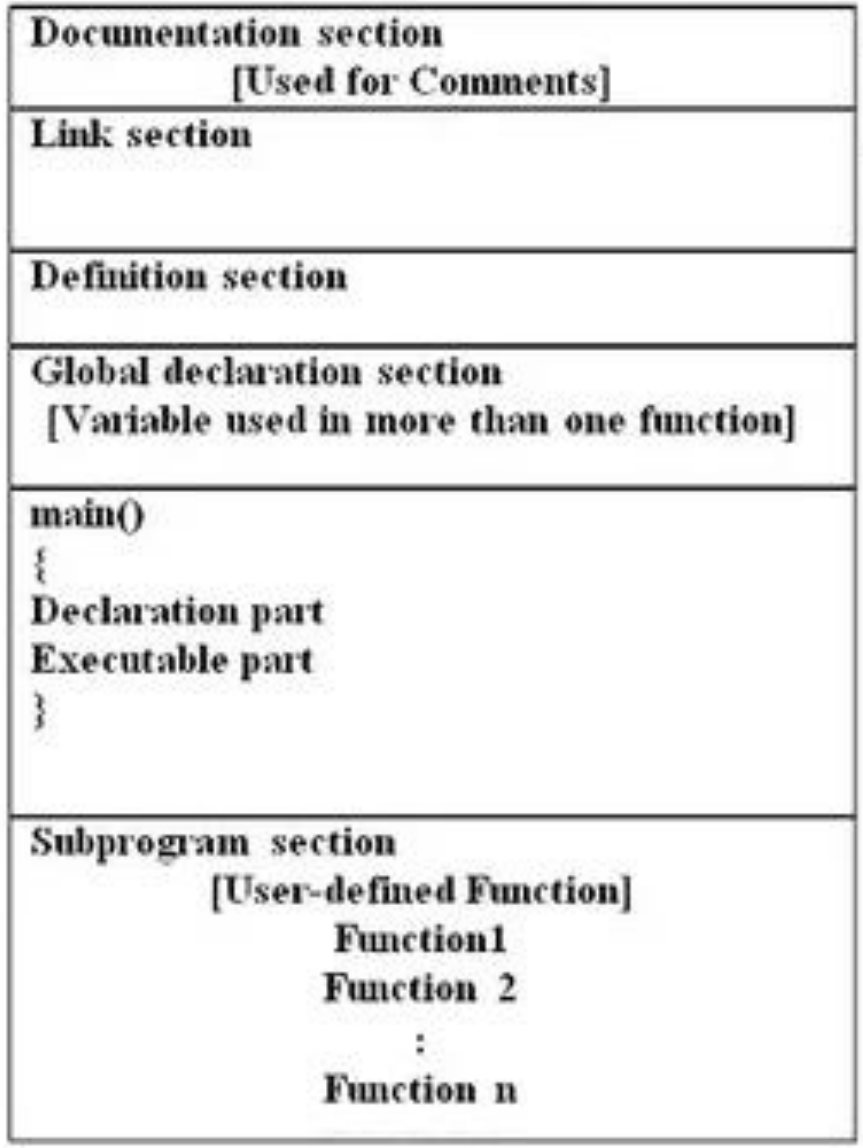
1A27-31/1A42-46

Lab Password: cslab768

Topics covered

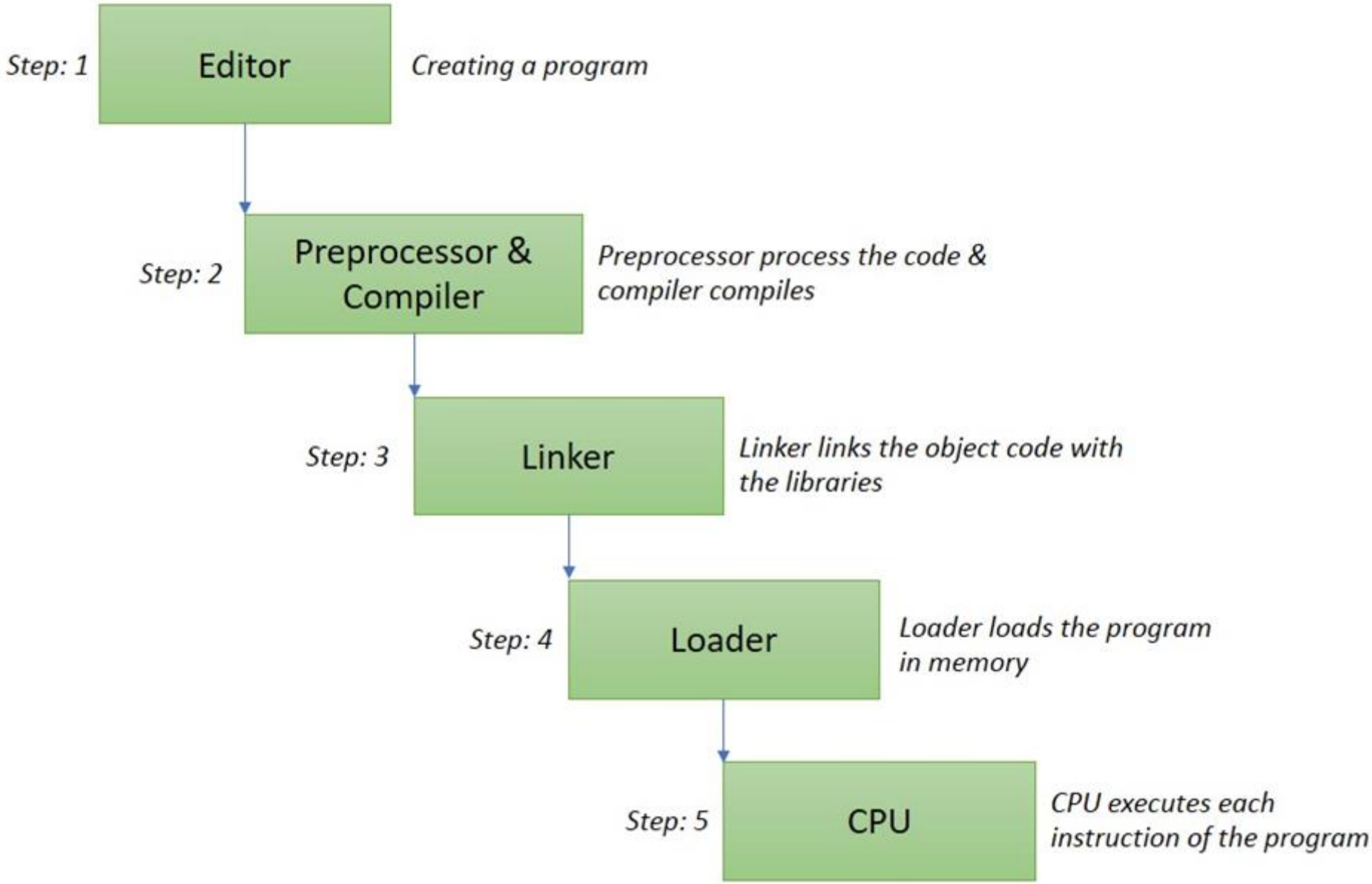
Structure of C Program, Life Cycle of Program from Source code to Executable, Compiling and Executing C Code, Keywords, Identifiers, Primitive Data types in C, variables, constants, input/output statements in C, operators, type conversion and type casting. Conditional branching statements, iterative statements, nested loops, break and continue statements.

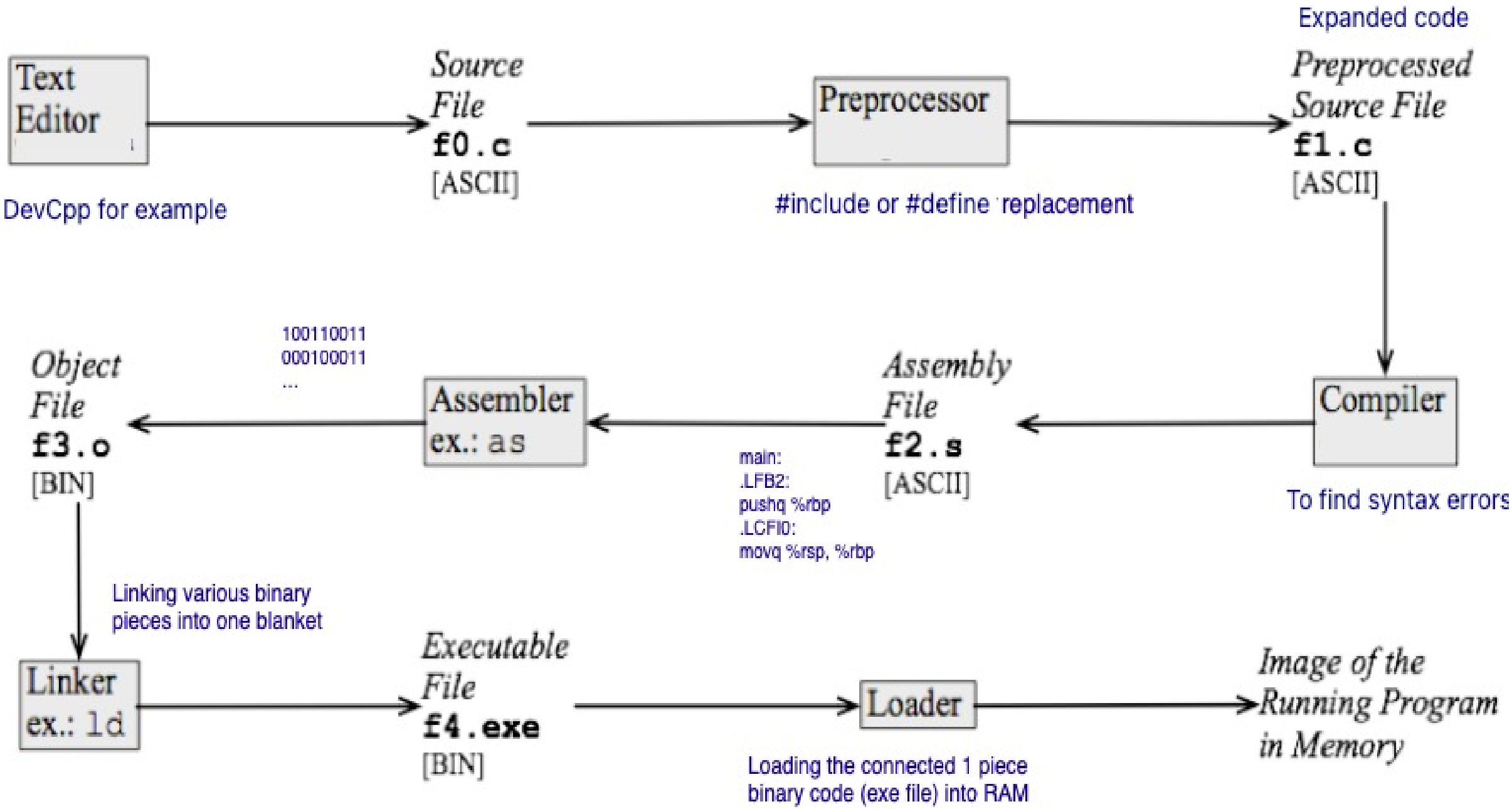
BASIC STRUCTURE OF A 'C' PROGRAM:



Example:

```
//Sample Prog Created by:Bsource

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

void fun();

int a=10;

void main()
{
printf("a value inside main(): %d",a);
fun();
}

void fun()
{
printf("\na value inside fun(): %d",a);
}
```


Step: 1
Editor
Creating a program
Step: 2
Preprocessor & Compiler
Preprocessor process the code & compiler compiles
Step: 3
Linker
Linker links the object code with the libraries
Step: 4
Loader
Loader loads the program in memory
Step: 5
CPU
CPU executes each instruction of the program


Text Editor
DevCpp for example
Source File
f0.c
[ASCII]
Preprocessor
#include or #define replacement
Expanded code
Preprocessed Source File
f1.c
[ASCII]
Compiler
To find syntax errors
Assembly File
f2.s
[ASCII]
main:
.LFB2:
pushq %rbp
.LCFI0:
movq %rsp, %rbp
Assembler
ex.: as
100110011
000100011
...
Object File
f3.o
[BIN]
Linking various binary pieces into one blanket
Linker
ex.: ld
Executable File
f4.exe
[BIN]
Loader
Loading the connected 1 piece binary code (exe file) into RAM
Image of the Running Program in Memory

C Tokens

- C Tokens are of 6 types, and they are classified as:

1. Identifiers,
2. Keywords,
3. Constants,
4. Operators,
5. Special Characters and
6. Strings.

Keywords or reserved words

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typeodef	
default	for	short	union	

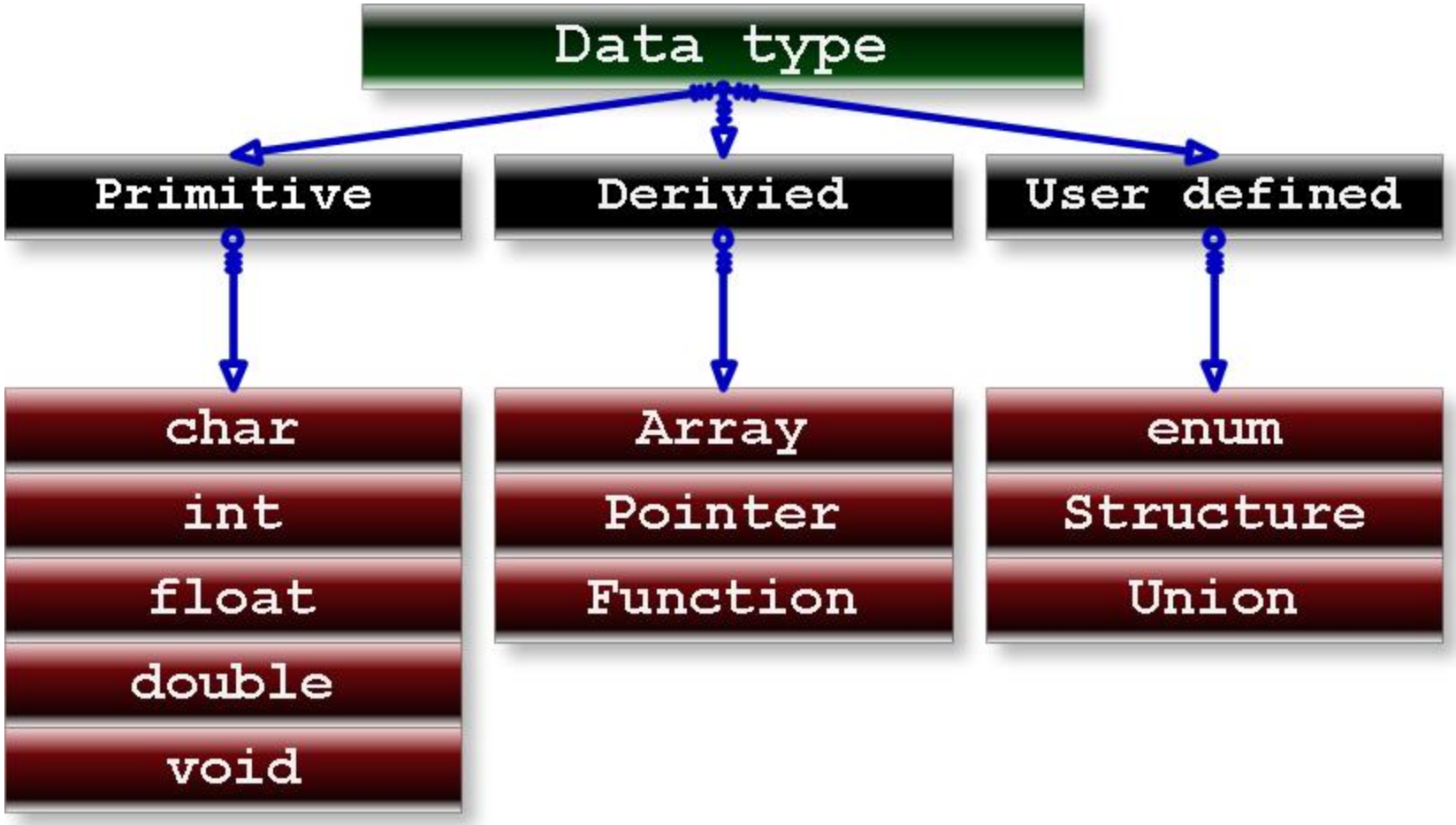
You cannot use them for variable names

Identifiers – variable names

- They must start with an alphabet
- May contain digits or alphabets.

```
int abc123, abc_123, abc;  //valid identifiers
```

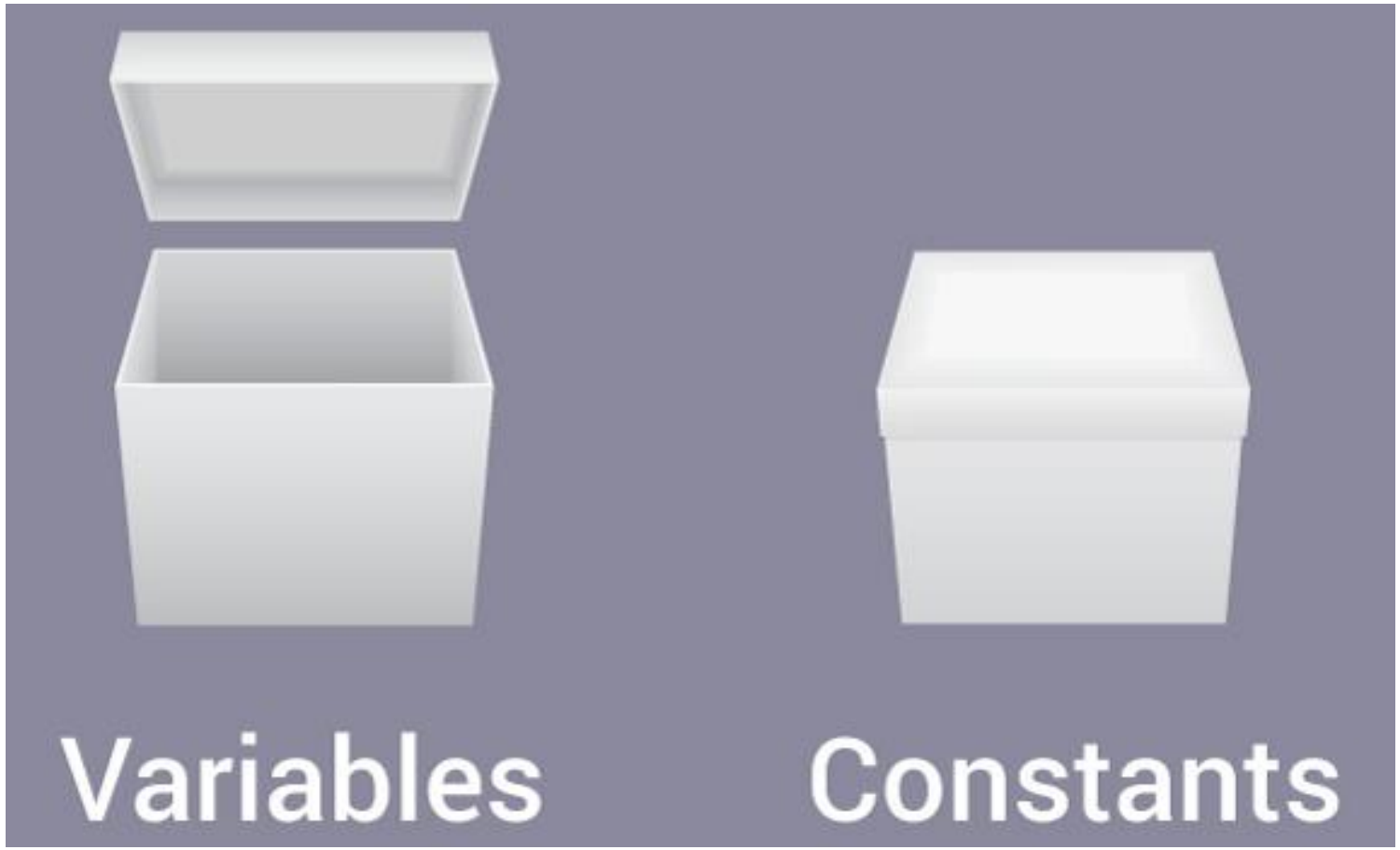
Datatypes in C



Primitive Data types in C

Keyword	Identifier (Format Specifier)	Size (Bytes)	Data Range
char	%c	1	-128 to +127
int	%d	4	-2^{31} to $+2^{31}$
float	%f	4	-3.4e38 to +3.4e38
double	%lf	8	-1.7e308 to +1.7e308
long int	%ld	8	-2^{63} to $+2^{63}$
unsigned int	%u	4	0 to 2^{32}
long double	%Lf	16	
unsigned char	%c	1	0 to 255

Variables and constants



I/O statements in C

Input: scanf(), gets()

Output: printf(), puts()

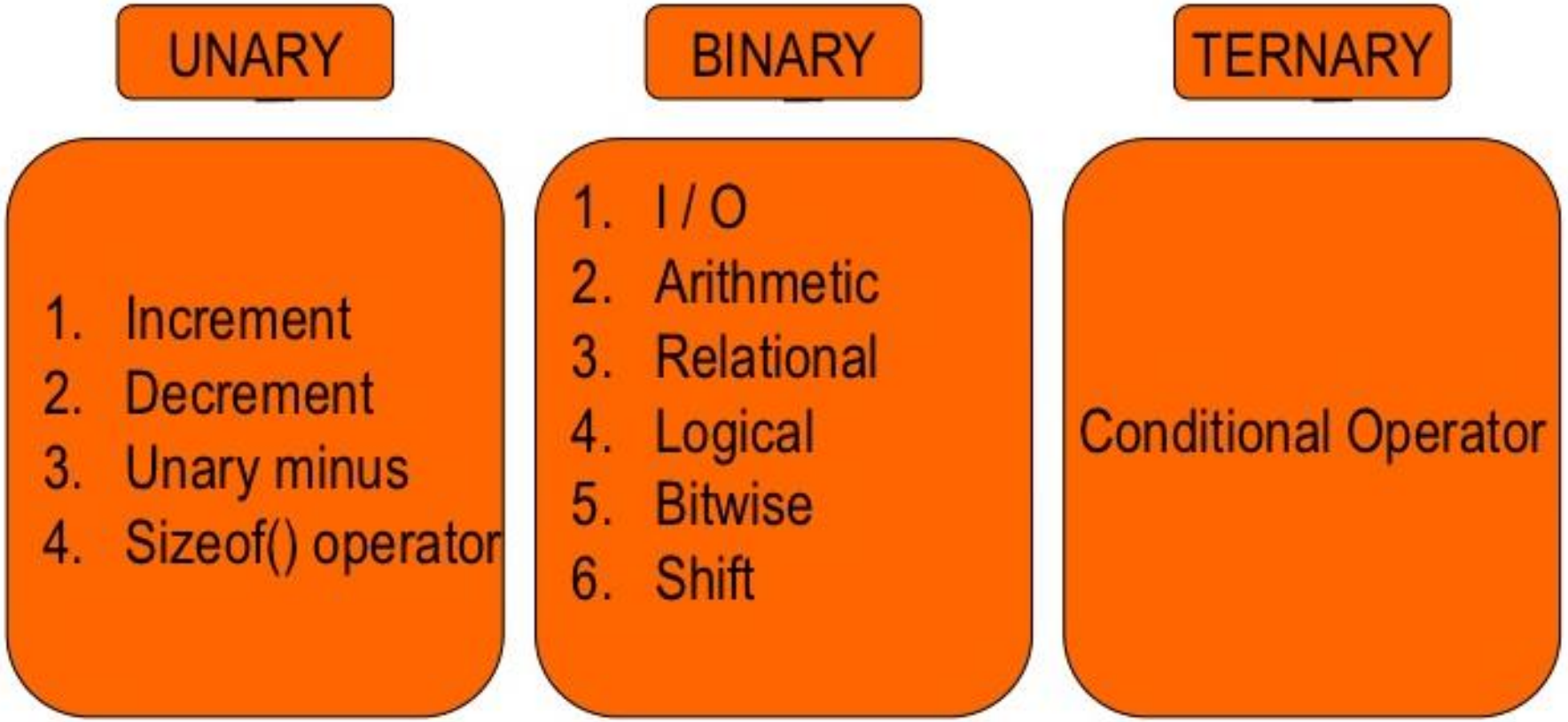
```
main(){
char s[20];  //string is array of char
puts("Enter a string: "); gets(s);
puts("Here is your string"); puts(s);
}
```

Operators in C

- Uniary (1 var) x = x+1; or x++;
- Binary (2 var) a = b*c;
- Ternary operator (3 var) ?:

```
main(){
int a=1,b=2,c;
c = a>b?a:b;  //condition?option1:option2
printf("c = %d\n",c);
}  //output is 2
```

Operators – 3 types



Operator Precedence of Arithmetic Operators

In decreasing order of priority

1. Parentheses :: ( )
2. Unary minus :: –5
3. Multiplication, Division, and Modulus
4. Addition and Subtraction

For operators of the *same priority*, evaluation is from *left to right* as they appear.

Parenthesis may be used to change the precedence of operator evaluation.

EXAMPLES:

a + b * c – d / e	a + (b * c) – (d / e)
a * – b + d % e – f	a * (– b) + (d % e) – f
a – b + c + d	(((a – b) + c) + d)
x * y * z	((x * y) * z)
a + b + c * d * e	(a + b) + ((c * d) * e)

Integer, Real, and Mixed-mode Arithmetic

INTEGER ARITHMETIC

- When the operands in an arithmetic expression are integers, the expression is called *integer expression*, and the operation is called *integer arithmetic*.
- Integer arithmetic always yields integer values.

  For example:

  25 / 10 evaluates to 2

REAL ARITHMETIC

- Arithmetic operations involving only real or floating-point operands.
- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.

  1.0 / 3.0 * 3.0 will have the value 0.99999 and not 1.0
- The modulus operator cannot be used with real operands.

MIXED-MODE ARITHMETIC

- When one of the operands is integer and the other is real, the expression is called a *mixed-mode* arithmetic expression.
- If either operand is of the real type, then only real arithmetic is performed, and the result is a real number.

  25 / 10 evaluates to 2

  25 / 10.0 evaluates to 2.5

Similar code – different results !!

```
1. int a = 10, b = 4, c;
2. float x;
3. c = a / b;
4. x = a / b;
```

- The value of c will be 2
- The value of x will be 2.0
- But we want 2.5 to be stored in x

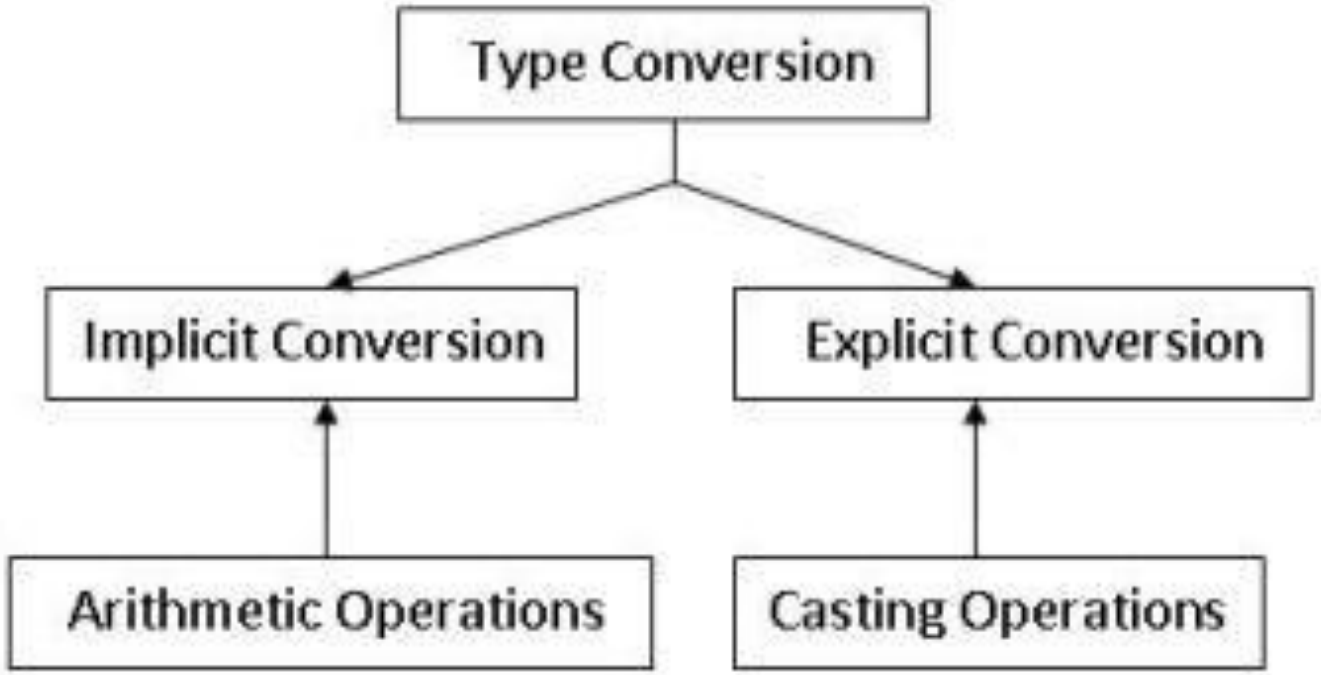
Solution: Typecasting

```
int a=10, b=4, c;
float x;
c = a / b;
x = ((float) a) / b;
```

- Changing the type of a variable during its use
- General form

  (type_name) variable_name
- Example:

  x = ((float) a) / b;
- Now x will store 2.5 (type of a is considered to be float for this operation only, now it is a mixed-mode expression, so real values are generated)



Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

Example: Finding Average of 2 Integers

Wrong program !! Why?

```
int a, b;
float avg;
scanf("%d%d", &a, &b);
avg = (a + b)/2;
printf("%f\n", avg);
```

```
int a, b;
float avg;
scanf("%d%d", &a, &b);
avg = ((float) (a + b))/2;
printf("%f\n", avg);
```

Correct programs

```
int a, b;
float avg;
scanf("%d%d", &a, &b);
avg = (a + b) / 2.0;
printf("%f\n", avg);
```

Restrictions on Typecasting

- Not everything can be typecast to anything
- **float/double** should not be typecast to **int** (as an int cannot store everything a float/double can store)
- **int** should not be typecast to **char** (same reason)

Integer and double are default

```
main(){
printf("%d\n",sizeof('x'));  //int
 printf("%d\n",sizeof(55));    //int
 printf("%d\n",sizeof(55.0));  //double
 }

//Output will be 4 4 8 because ‘x’ will become integer for its
   ASCII value
```

Type conversion - implicit

```
#include<stdio.h>
main(){
//implicit typecasting to float
float f = 10.0;
printf("\n f = %f",f);
printf("\nsize of 10.0 = %d",sizeof(10.0));
printf("\nsize of f = %d",sizeof(f));
}
```

//output: f = 10.0000 size of 10.0 = 8 size of f = 4

Type conversion - explicit

```
main(){
//explicit typecasting to float
float f = (float)10.0;
printf("\n value of  = %f",f);
}
```

Doing More Complex Mathematical Operations

- C provides some mathematical functions to use in the math library
  - Can be used to perform common mathematical calculations
  - Two steps needed:
    (1) Must include a special header file
    `#include <math.h>`
    (2) Must tell the compiler to link the math library: gcc <program name> –lm

- Example

```
printf ("%f", sqrt(900.0));
```

  - Calls function sqrt, which returns the square root of its argument

- Return values of math functions are of type double
- Arguments may be constants, variables, or expressions

Math Library Functions

double acos(double x)	– Compute arc cosine of x.
double asin(double x)	– Compute arc sine of x.
double atan(double x)	– Compute arc tangent of x.
double atan2(double y, double x)	– Compute arc tangent of y/x.
double cos(double x)	– Compute cosine of angle in radians.
double cosh(double x)	– Compute the hyperbolic cosine of x.
double sin(double x)	– Compute sine of angle in radians.
double sinh(double x)	– Compute the hyperbolic sine of x.
double tan(double x)	– Compute tangent of angle in radians.
double tanh(double x)	– Compute the hyperbolic tangent of x.

Math Library Functions

double ceil(double x)	– Get smallest integral value that exceeds x.
double floor(double x)	– Get largest integral value less than x.
double exp(double x)	– Compute exponential of x.
double fabs (double x)	– Compute absolute value of x.
double log(double x)	– Compute log to the base e of x.
double log10 (double x)	– Compute log to the base 10 of x.
double pow (double x, double y)	– Compute x raised to the power y.
double sqrt(double x)	– Compute the square root of x.

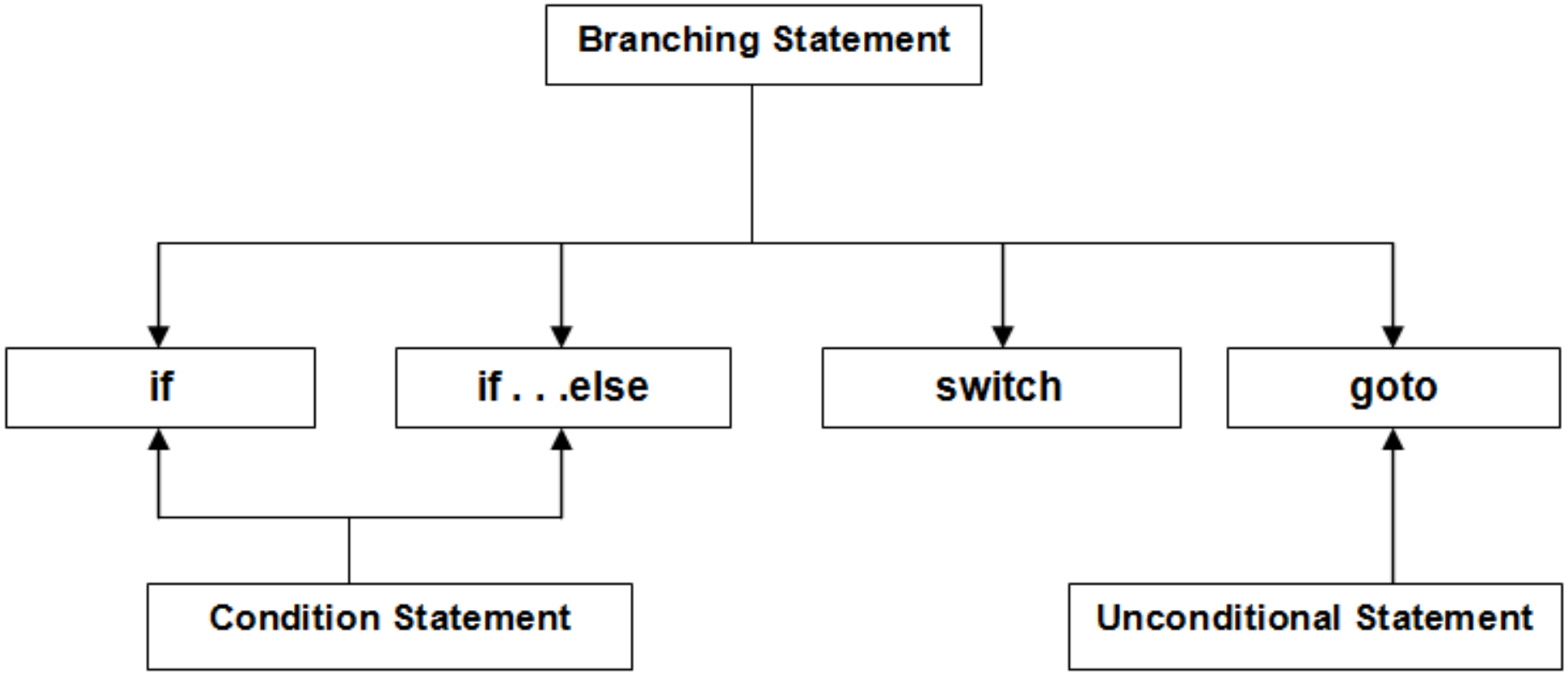
Computing distance between two points

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int x1, y1, x2, y2;
    double dist;
    printf("Enter coordinates of first point: ");
    scanf("%d%d", &x1, &y1);
    printf("Enter coordinates of second point: ");
    scanf("%d%d", &x2, &y2);
    dist = sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
    printf("Distance = %lf\n", dist);
    return 0;
}
```

Practice Problems

- Read in three integers and print their average
- Read in four integers a, b, c, d. Compute and print the value of the expression a+b/c/d*10*5-b+20*d/c
  - Explain to yourself the value printed based on precedence of operators taught
  - Repeat by putting parentheses around different parts (you choose) and first do by hand what should be printed, and then run the program to verify if you got it right
  - Repeat similar thing for the expression a&&b||c&&d>a||c<=b
- Read in the coordinates (real numbers) of three points in 2-d plane, and print the area of the triangle formed by them

Conditional branching statements

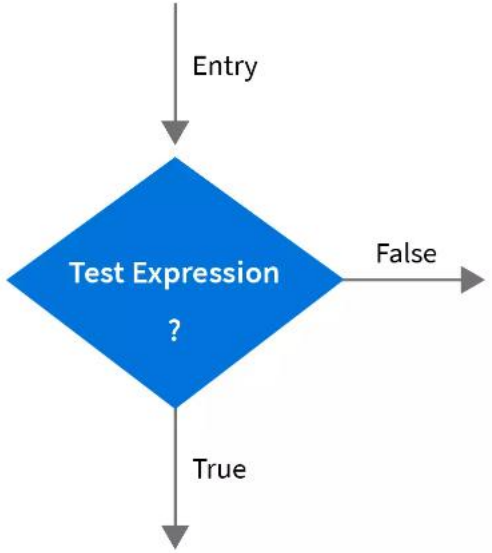


if

Types of If Statement

if statement may be implemented in different forms depending on the complexity of testing conditions to be evaluated.

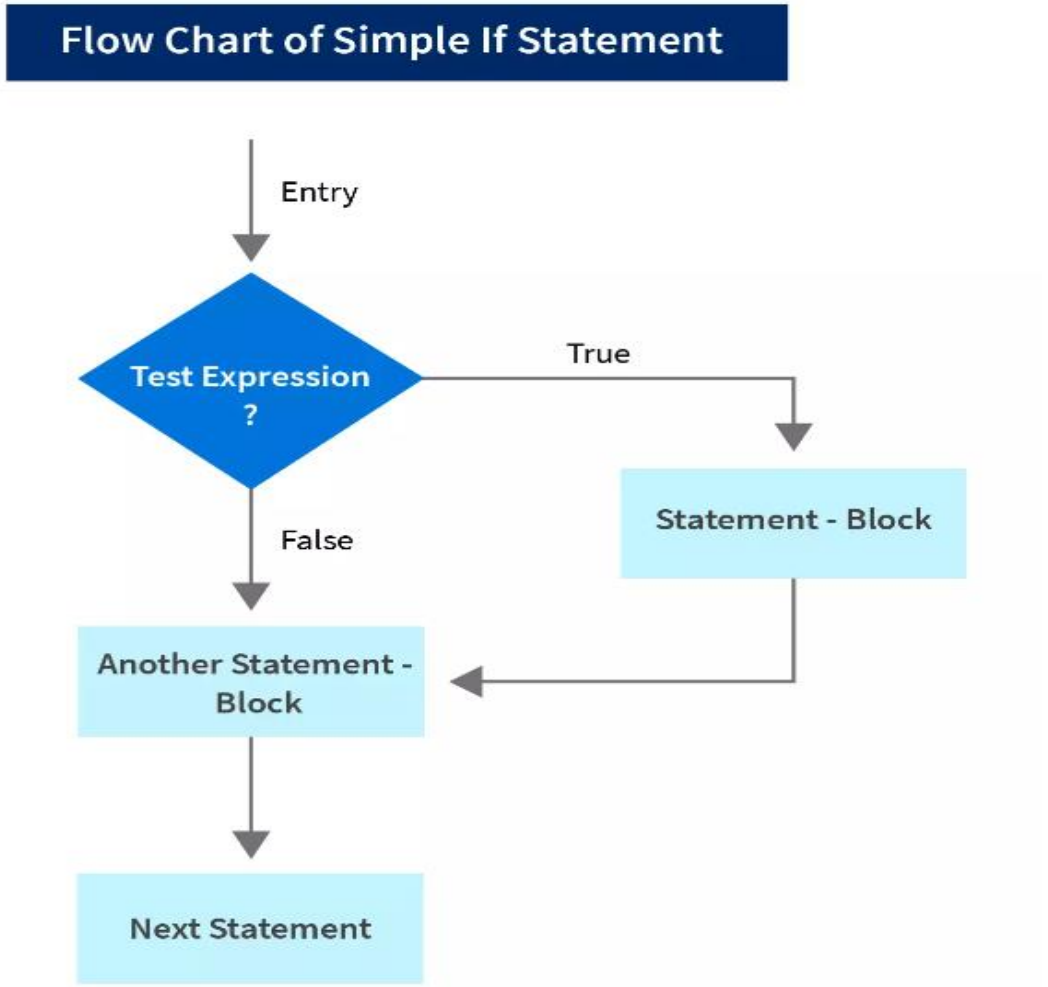
- Simple if Statement
- if-else Statement
- Nested if-else Statement
- else-if Ladder



Simple If

```
if(gender is Female)
   Person is Female
```

```
if(age is more than 60)
   person is retired
```



If statement

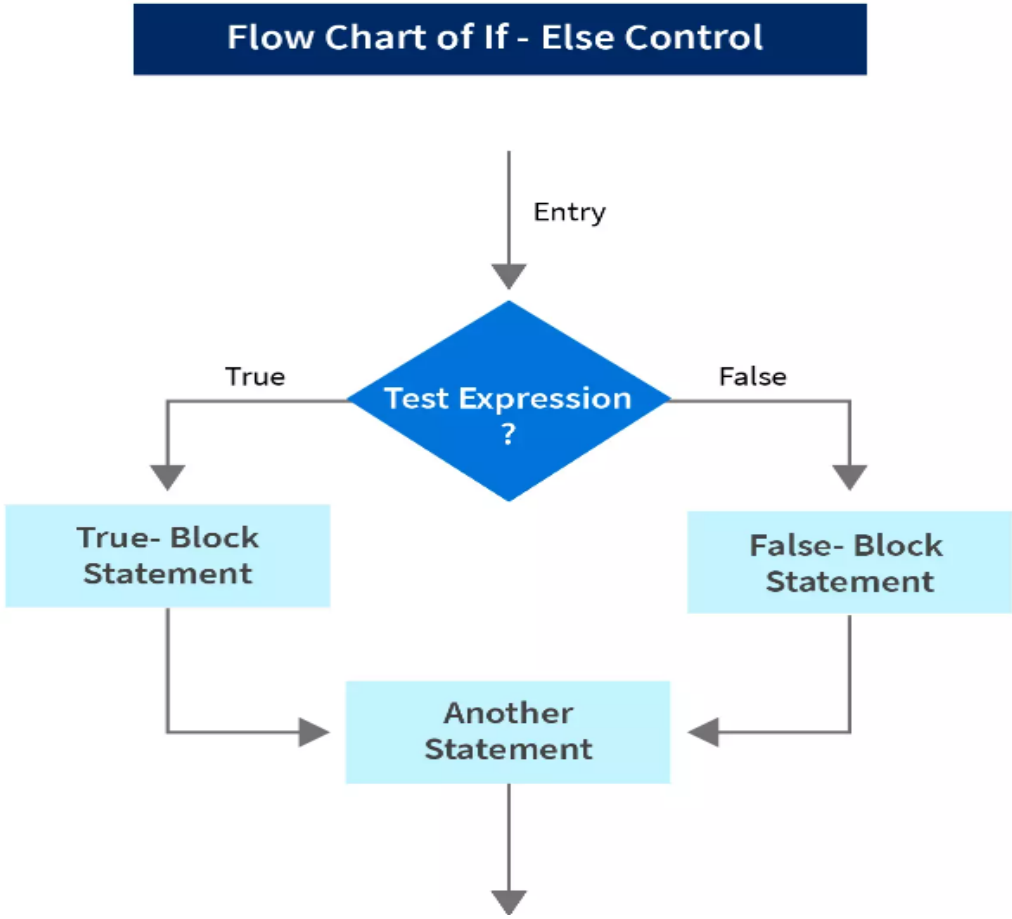
```
main(){
int a=10;
if(a<20) {
 printf("a<20\n");
}
if(a>20) {
 printf("a>20\n");
}}  //output is a<20
```

If-else

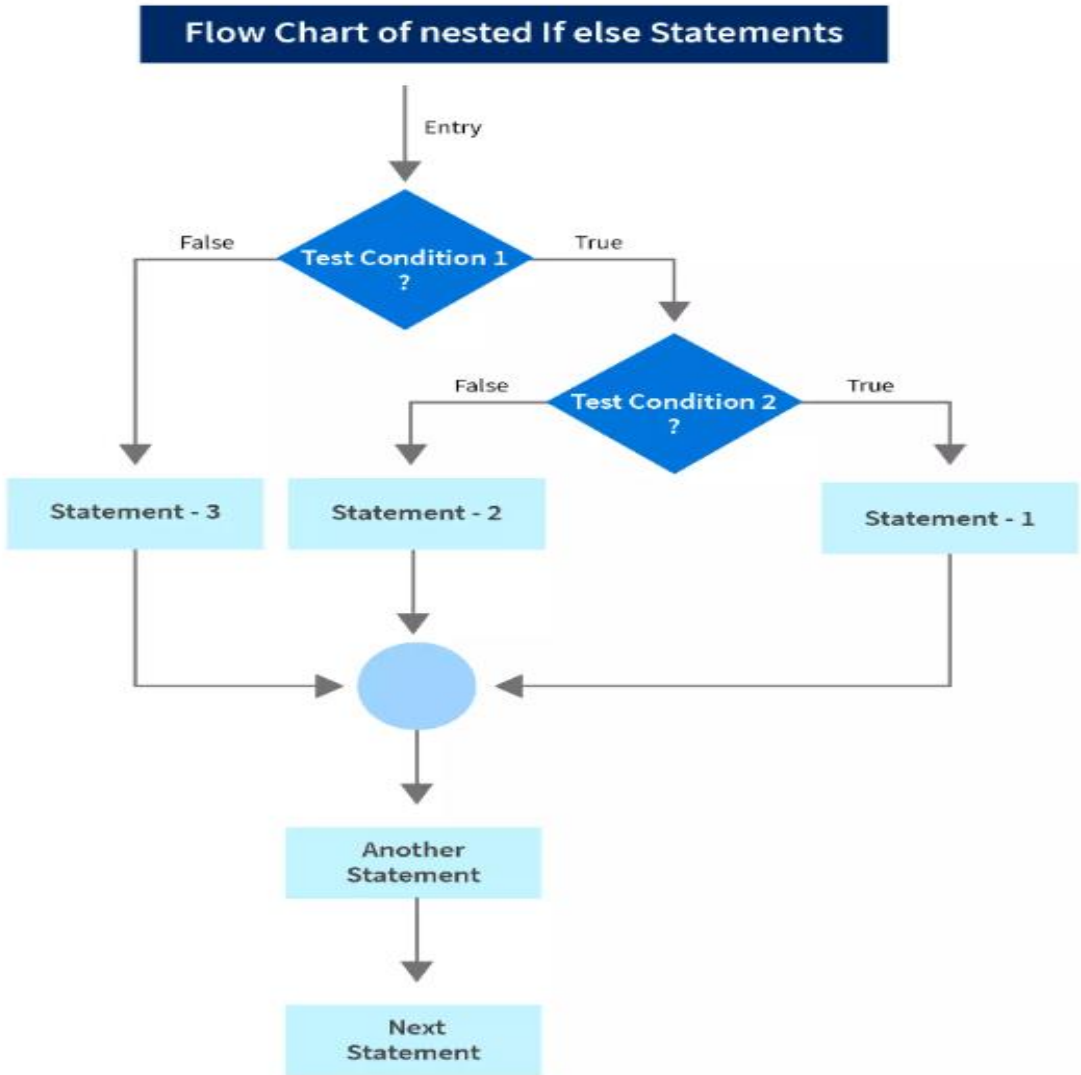
```
main(){
int a=10;
if(a<20) {
 printf("a<20\n");
}
else {
 printf("a>20\n");
}}  //output is a<20
```

If-else

```
if(test expression) {
  true-block statement(s)
}
else {
  false-block statement(s)
}
  another-statement
```

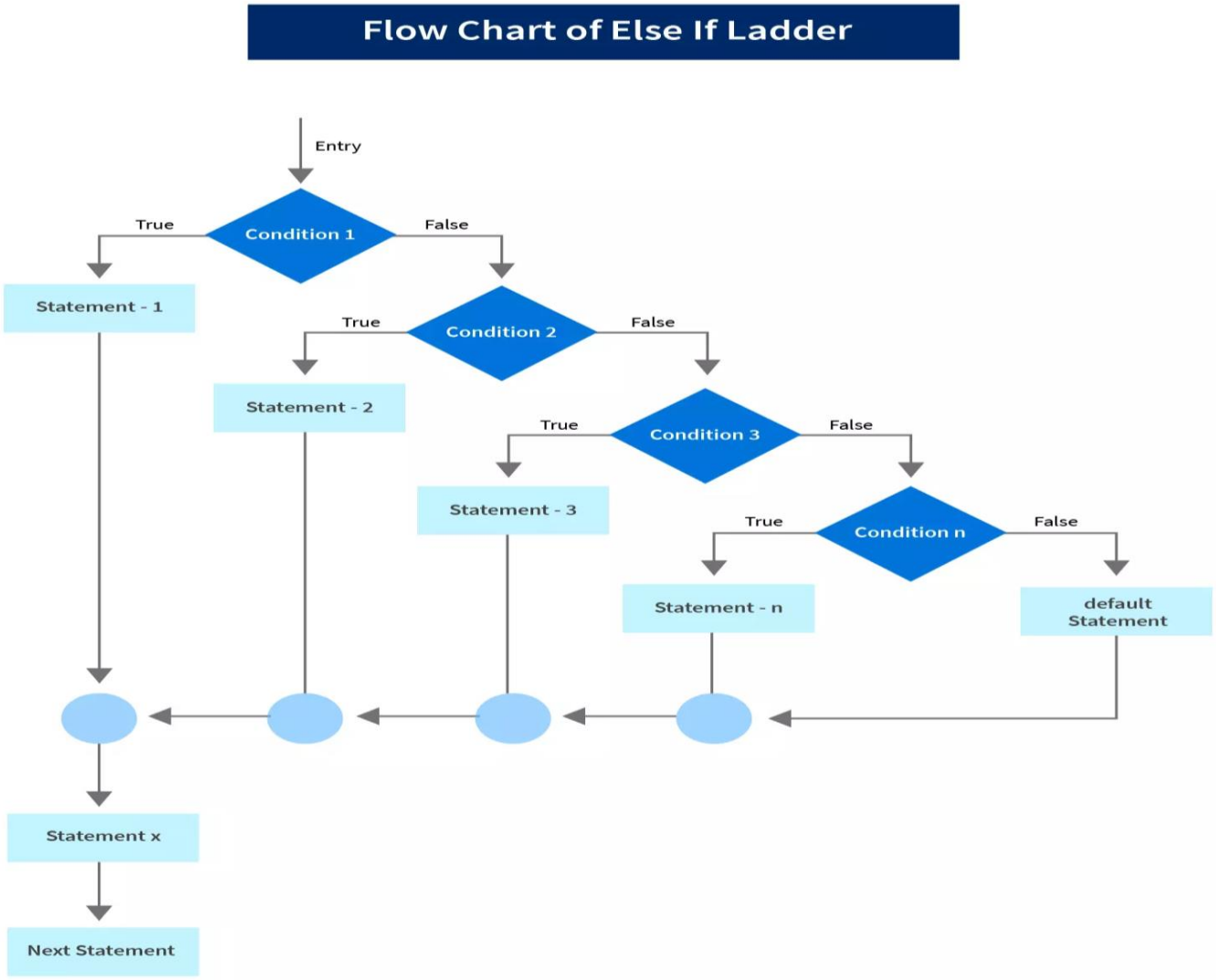


Nested if else



Else-if Ladder:

```
if(test expression) {
  true-block statement
}
else if(test expression){
  block of statement
}
else if(test expression){
 block of statement
}
else {
  false-block statement
}
```



Important Points Need to Remember

- Never put semicolon just after the if(expression).
- A non-zero value is considered as true and a zero(0) value is considered as false in C.
- We can use more than one condition inside the if statement using the logical operator.
- We should always use braces on separate lines to identify a block of statements.
- We should always align the opening and closing braces.
- Do not ignore placing parentheses for the if condition/expression.
- Be aware of dangling else statements.
- Avoid using operands that have side effects in a logical binary expression such as (a-- && ++b). The second operand may not be evaluated in any case.

examples

```
if(age > 39)
   printf("You are so old!\n");
```

The if statement

Fundamental means of *flow control*

How we will make decisions

Boolean expressions

The actual determination of the decision

```
age > 39
c == 0
l <= 0
(age >= 18) && (age < 65)
```

Basic Boolean Expressions

true
false
age < 18
divisor == 0
size > 1000000
ch == 'X'

Some operators:

<	Less than
<=	Less than or equal to
==	Equal
!=	Not equal
>=	Greater than or equal to
>	Greater than

Important: The test for equality is ==, not =. This is the most common error in a C program.

Example if statements

<	Less than
<=	Less than or equal to
==	Equal
!=	Not equal
>=	Greater than or equal to
>	Greater than

```
if(age < 18)
   printf("Too young to vote!\n");
```

```
if(area == 0)
   printf("The plot is empty\n");
else
   printf("The plot has an area of %.1f\n", area);
```

```
if(val < 0)
   printf("Negative input is not allowed\n");
else if(val == 0)
   printf("A value of zero is not allowed\n");
else
   printf("The reciprocal is %.2f\n", 1.0 / val);
```

Note the indentation

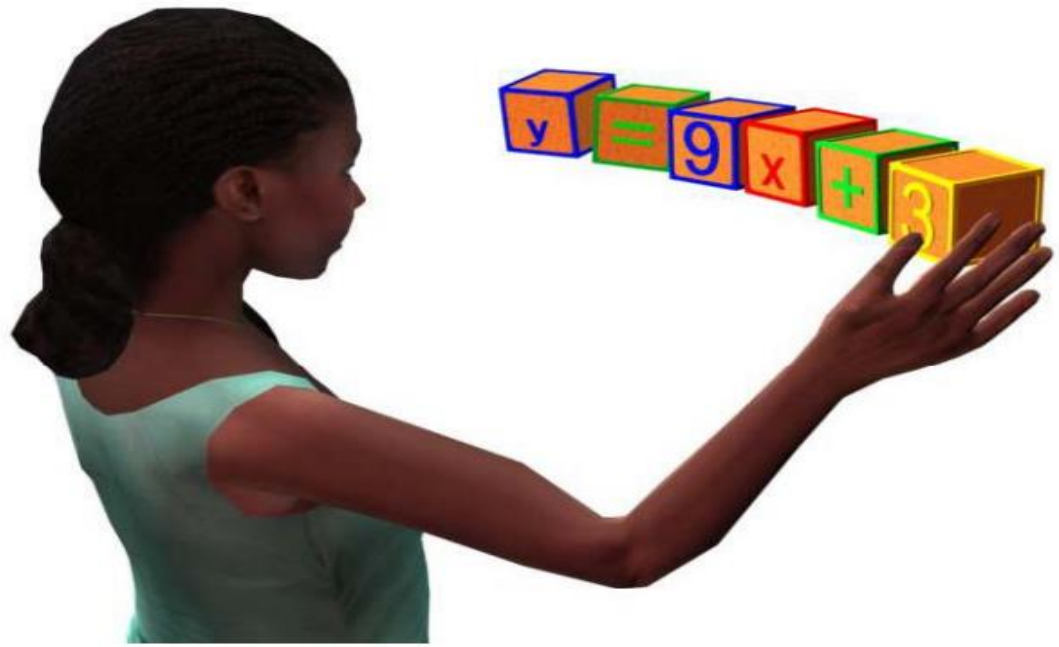
Blocks

Single Statement

```
printf("This is a statement\n");
```

Block

```
{
    printf("All items in a curly brace\n");
    printf("as if there are one statement");
    printf("They are executed sequentially");
}
```



Where is this useful?

```
if(value > 0)
{
    result = 1.0 / value;
    printf("Result = %f\n", result);
}
```

If the expression is true, all of the statements in the block are executed

Where is this useful?

Will these two sections of code work differently?

```
if(value > 0)
{
    result = 1.0 / value;
    printf("Result = %f\n", result);
}
```

```
if(value > 0)
    result = 1.0 / value;
    printf("Result = %f\n", result);
```

Nested Blocks

What does this do?

```
if(bobsAge != suesAge)  /* != means "not equal" */
{
    printf("Bob and Sue are different ages\n");
    if(bobsAge > suesAge)
    {
        printf("In fact, Bob is older than Sue\n");
        if((bobsAge - 20) > suesAge)
        {
            printf("Wow, Bob is more than 20 years older\n");
        }
    }
}
```

Importance of indentation

See how much harder this is to read?

```
if(bobsAge != suesAge)  /* != means "not equal" */
{
printf("Bob and Sue are different ages\n");
if(bobsAge > suesAge)
{
printf("In fact, Bob is older than Sue\n");
if((bobsAge - 20) > suesAge)
{
printf("Wow, Bob is more than 20 years older\n");
}
}
}
```

More Examples

- char myChar = ‘A’;
  - The value of myChar==‘Q’ is false (0)

- Be careful when using floating point equality comparisons, especially with zero, e.g. myFloat==0

Suppose?

What if I want to know if a value is in a range?

Test for: $100 \leq L \leq 1000$?

You can't do...

This code is WRONG and will fail.

```
if(100 <= L <= 1000)
{
    printf("Value is in range...\n");
}
```

Why this fails…


```
if((100 <= L) <= 1000)
{
    printf("Value is in range…\n");
}
```

Suppose L is 5000. Then 100 <= L is true, so (100 <= L) evaluates to true, which, in C, is a 1. Then it tests 1 <= 1000, which also returns true, even though you expected a false.

Compound Expressions

- Want to check whether -3 <= B <= -1
  - Since B = -2, answer should be True (1)
- But in C, the expression is evaluated as
  - ((-3 <= B) <= -1) (<= is left associative)
  - (-3 <= B) is true (1)
  - (1 <= -1) is false (0)
  - Therefore, answer is 0!

Compound Expressions

- Solution (not in C): (-3<=B) and (B<=-1)

- In C: (-3<=B) && (B<=-1)
- Logical Operators
  - And: &&
  - Or: ||
  - Not: !

Compound Expressions

```
#include <stdio.h>

int main()
{
    const int A=2, B = -2;

    printf("Value of A is %d\n", A);
    printf("0 <= A <= 5?: Answer=%d\n", (0<=A) && (A<=5));

    printf("Value of B is %d\n", B);
    printf("-3 <= B <= -1?: Answer=%d\n", (-3<=B) && (B<=-1));
}
```

comparators comparators

```
1. #include<stdio.h>
2. int main(){
3.     const int int3 = 3, int8 = 8;
4.     printf("\nint3 comparators comparators\n ");
5.     printf("int3 == int8: %d\n",(int3 == int8));
6.     printf("int3!=int8:%d\n",int3!=int8);
7.     printf("int3 < 3: %d\n",(int3 < 3));
8.     printf("int3 <= 3: %d\n",(int3 <= 3));
9.     printf("int3 > 3: %d\n",(int3 > 3));
10.    printf("int3 >= 3: %d\n",(int3 >= 3));
11.}
```

Precedence & Associativity

```
int main() {
    int A = 4, B = 2;

    printf("Answer is %d\n", A + B > 5 && (A = 0) < 1 > A + B - 2);
}
```

Relational operators have precedence and associativity (just like arithmetic operators)

Use ( ) when in doubt

```
A = 4, B = 2;
A + B > 5 && (A = 0) < 1 > A + B - 2
```

(((A + B) > 5) && (((A=0) < 1) > ((A + B) − 2)))

((6 > 5) && (((A=0) < 1) > ((A + B) − 2)))

(1 && ((0 < 1) > ((A + B) − 2)))

(1 && (1 > (2 − 2)))

(1 && (1 > 0))

(1 && 1)

Answer: 1

Precedence: +/- > < &&

Associativity

"=" is right associative

Example: X=Y=5

right associative: X = (Y=5)

expression Y=5 returns

value 5: X = 5

Logical Operators

- Unary and Binary Operators

  ! ➔ Logical NOT, logical negation (True if the operand is False.)

  && ➔ Logical AND (True if both the operands are True.)

  || ➔ Logical OR (True if either one of the operands is True.)

X	!X
FALSE	TRUE
TRUE	FALSE

X	Y	X && Y	X || Y
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

```
        int x = 20; int y=3;float
        a=20.3;

if((x>y) && (x>a))    /* FALSE */
        printf("X is largest.");

if((x>y) || (x>a))    /* TRUE */
    printf("X is not smallest.");

if(!(x==y))    /* TRUE */
    printf("X is not same as Y.");

if(x!=y)       /* TRUE */
    printf("X is not same as Y.");
```

Control Statements

Statement takes more than one branches based upon a **condition test** comprising of relational and/or logical (may be arithmetic) operators.

Some set of statements are being executed **iteratively** until a **condition test** comprising of relational and/or logical (may be arithmetic) operators are not being satisfied.

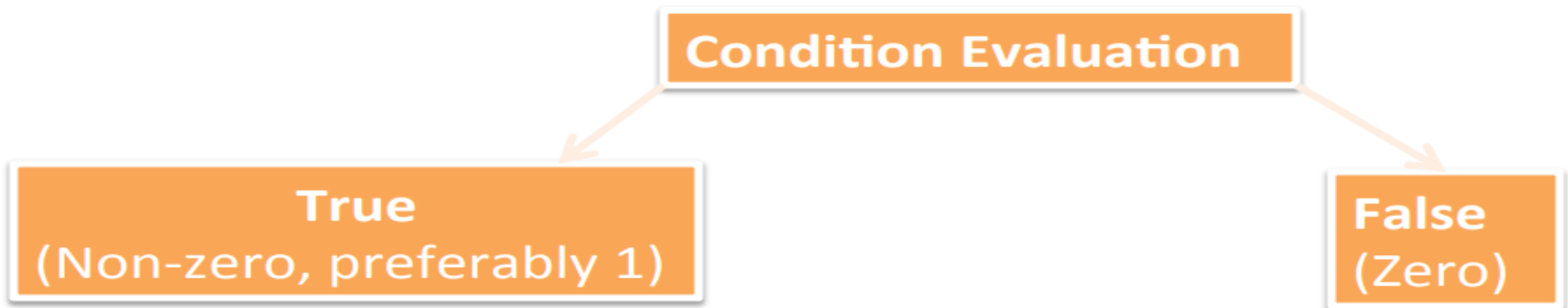
Conditions

- Using relational operators.
  - Four relation operators: <, <=, >, >=
  - Two equality operations: ==, !=

- Using logical operators / connectives.
  - Two logical connectives: &&, | |
  - Unary negation operator: !

Condition Tests

```
if(count <= 100)          /* Relational */
if((math+phys+chem)/3 >= 60) /* Arithmetic, Relational */
if((sex==‘M’) && (age>=21))  /* Relational, Logical */
if((marks>=80) && (marks<90) )  /* Relational, Logical */
if((balance>5000) | | (no_of_trans>25))  /* Relational,
  Logical */
if(! (grade==‘A’))              /* Relational, Logical */
```



Operator confusion

Equality (==) and Assignment (=) Operators

Better is avoid.

```
int age=20;
        if ( age > 18 )  /* Logical Operator; Evaluated as TRUE */
          printf( "You are not a minor!\n" );

        if ( age >= 18 ) /* Logical Operator; Evaluated as TRUE */
          printf( "You are not a minor!\n" );

        if ( age == 20 ) /* Logical Operator; Evaluated as TRUE */
          printf( "You are not a minor!\n" );

        if ( age = 18 )  /* Arithmetic Operator; Evaluated as TRUE */
          printf( "You are not a minor!\n" );

        if ( age = 17 ) /* Arithmetic Operator; Evaluated as TRUE */
          printf( "You are a minor!\n" );
```

Value of age will be 18

Value of age will be 17

There will be no syntax error.

These statements are not logically correct!!!

Example 3

- Even /Odd

Operator confusion

Equality (==) and Assignment (=) Operators

```
#include <stdio.h>
int main()
{
    int x,y;
    scanf("%d",&x);
    y=x%2;      /* y will be 1 or zero based on value entered
and stored as x */
    if(y=1) { /* y will be assigned with 1, condition will be
evaluated as TRUE */
        printf("Entered number is odd.");
    } else {
        printf("Entered number is even.");
    }

    return 0;
}
```

Unary Operator

- Increment (++) Operation means i = i + 1;
  - Prefix operation (++i) or Postfix operation (i++)
- Decrement (--) Operation means i = i - 1;
  - Prefix operation (--i) or Postfix operation (i--)
- Precedence
  - Prefix operation : First increment / decrement and then used in evaluation
  - Postfix operation : Increment / decrement operation after being used in evaluation
- Example

```
int t, m=1;
t=++m;
```

m=2
t=2

```
int t,m=1;
t=m++;
```

m=2
t=1

More Examples on Unary Operator

Initial values :: a = 10; b = 20

```
x = 50 + ++a;
```

a = 11, x = 61

Initial values :: a = 10; b = 20;

```
x = 50 + a++;
```

x = 60, a = 11

Initial values :: a = 10; b = 20;

```
x = a++ + --b;
```

b = 19, x = 29, a = 11

Initial values :: a = 10; b = 20;

```
x = a++ - ++a;
```

Undefined value (implementation dependent)

Shortcuts in Assignment Statements

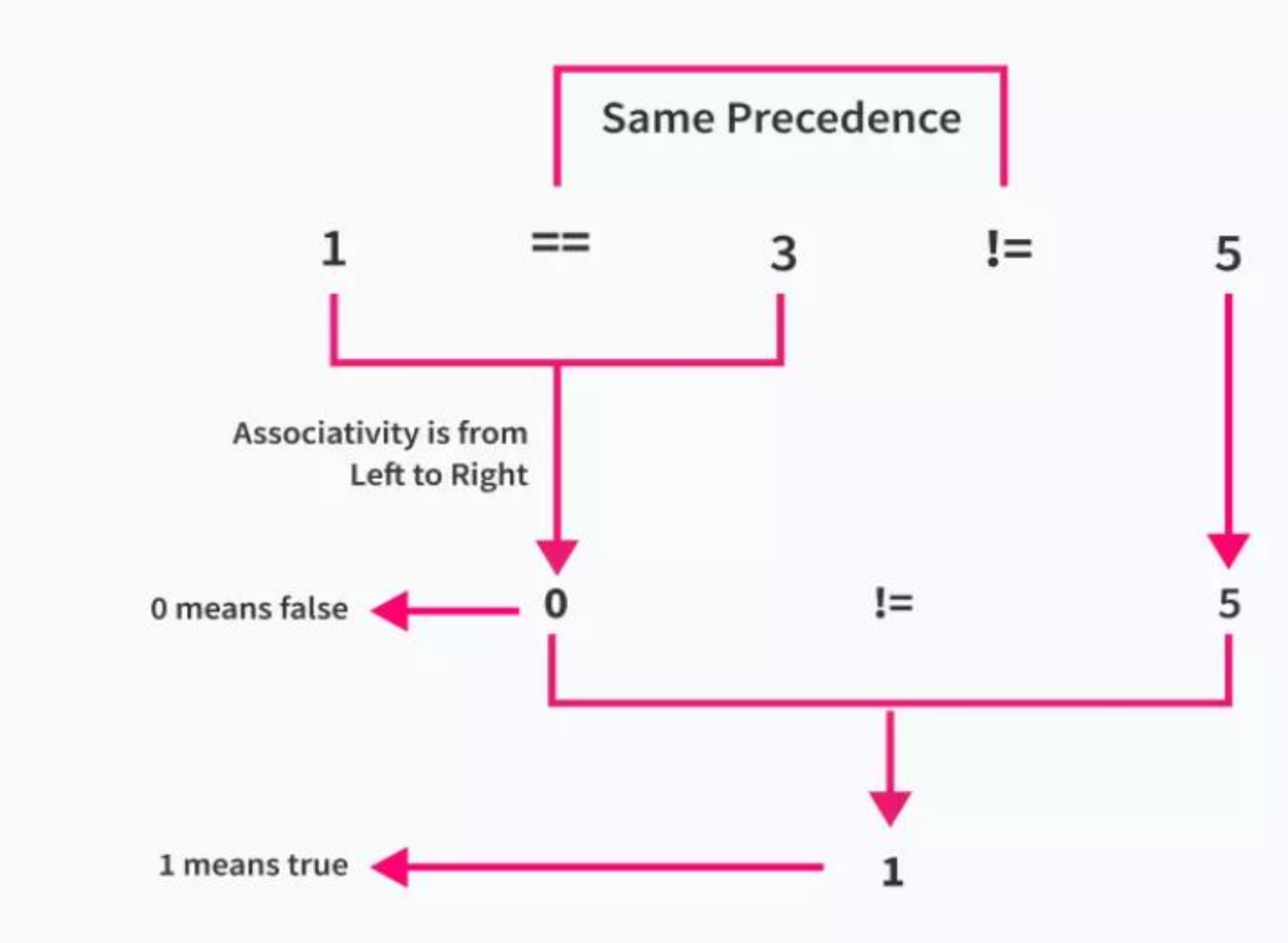
- A+=C → A = A+C
- A-=B → A = A-B
- A*=D → A = A*D
- A/=E → A = A/E

- You should refer to the C operator precedence and associative table.
- Or just use parentheses whenever you're unsure about precedence and associativity

Operator	*Description*	*Associativity*
() [] . ->>> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to and not equal to	left to right
&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
|	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
||	Logical OR	left to right
? :	Ternary operator	right to left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	Comma operator	left to right

Operator Precedence and Associativity in C

- printf("%d", 1 == 3 != 5);



Operator Precedence and Associativity in C

```
#include <stdio.h>

void main()
{
   int a = 5;

   int ans = ++a * (3 + 8) % 35 - 28 / 7;

   printf("%d", ans);
}
```

Precedence and Associativity of Postfix ++ and Prefix ++

- The precedence of postfix ++ is more than that of prefix ++. The associativity of postfix ++ is left to right, while the associativity of prefix ++ is right to left.
- #include <stdio.h>

```
1.  void main()
2.  {
3.      int a = 1, ans;
4.
5.      ans = a++ + ++a;
6.
7.      printf("%d", ans);
8.  }
```

Input

Conversion Character	Data Item meaning
c	Single charcater
d	Decimal integer
e	Floating point value
f	Floating point value
g	Floating point value
h	Short int
i	Decimal/hexadecimal/octal integer
o	Octal integer
s	String
u	Unsigned decimal integer
X	Hexadecimal integer

We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.

Example: scanf ("%3d %5d", &a, &b);

Output

```
printf ("control string",arg1,arg2, …, argn);
```

- Performs output to the standard output device (typically defined to be the screen).
- Control string refers to a string containing formatting information and data types of the arguments to be output;
- The arguments arg1, arg2, … represent the individual output data items.
- The conversion characters are the same as in scanf.

```
int size,a,b;
float length;
scanf ("%d", &size) ;
scanf ("%f", &length) ;
scanf ("%d  %d", &a, &b);
```

```
printf("%d",size);
printf("%f",length);
printf("%d %d",a,b);
```

Formatted Output

```
float a=3.0, b=7.0;
printf("%f %f %f %f",a,b,a+b,sqrt(a+b));
3.000000 7.000000 10.000000  3.162278
```

Total Space

Will be written exactly.

```
printf("%4.2f %5.1f\na+b=%3.2f\tSquare
Root=%-6.3f",a,b,a+b,sqrt(a+b));
3.00  7.0
a+b=10.00    Square Root=3.162
```

After decimal place

Tab

Left Align

For integer, character and string, no decimal point.

Character I/O

```
char ch1;
scanf("%c",&ch1);        /* Reads a character */
printf("%c",ch1);        /* Prints a character */
ch1=getchar();         /* Reads a character */
putchar(ch1);     /* Prints a character */
```

```
char name[20];
scanf("%s",name);        /* Reads a string */
printf("%s",name);       /* Prints a string */
gets(name);            /* Reads a string */
puts(name);            /* Prints a string */
```

More on the char type

- Is actually stored as an integer internally
- Each character has an integer code associated with it (ASCII code value)
- Internally, storing a character means storing its integer code
- All operators that are allowed on int are allowed on char
  - 32 + ‘a’ will evaluate to 32 + 97 (the integer ascii code of the character ‘a’) = 129
  - Same for other operators
- Can switch on chars constants in switch, as they are integer constants

Another example

```
int a;
a = 'c' * 3 + 5;
printf("%d", a);
```

Will print 302 (99*3 + 5)
(ASCII code of 'c' = 99

```
char c = 'A';
printf("%c = %d", c, c);
```

Will print A = 65
(ASCII code of 'A' = 65)

Assigning char to int is fine. But other way round is dangerous, as size of int is larger

ASCII Code

- Each character is assigned a unique integer value (code) between 32 and 127
- The code of a character is represented by an 8-bit unit. Since an 8-bit unit can hold a total of 2^8=256 values and the computer character set is much smaller than that, some values of this 8-bit unit do not correspond to visible characters
- But never try to remember exact ASCII codes while programming. Use the facts that
  - C stores characters as integers
  - ASCII codes of some important characters are contiguous (digits, lowercase alphabets, uppercase alphabets)

Decimal	Hex	Binary	Character	Decimal	Hex	Binary	Character
32	20	00100000	SPACE	80	50	01010000	P
33	21	00100001	!	81	51	01010001	Q
34	22	00100010	"	82	52	01010010	R
35	23	00100011	#	83	53	01010011	S
36	24	00100100	$	84	54	01010100	T
37	25	00100101	%	85	55	01010101	U
38	26	00100110	&	86	56	01010110	V
39	27	00100111	'	87	57	01010111	W
40	28	00101000	(	88	58	01011000	X
41	29	00101001	)	89	59	01011001	Y
42	2a	00101010	*	90	5a	01011010	Z
43	2b	00101011	+	91	5b	01011011	[
44	2c	00101100	,	92	5c	01011100	\
45	2d	00101101	-	93	5d	01011101	]
46	2e	00101110	.	94	5e	01011110	^
47	2f	00101111	/	95	5f	01011111	_
48	30	00110000	0	96	60	01100000	`
49	31	00110001	1	97	61	01100001	a
50	32	00110010	2	98	62	01100010	b

51	33	00110011	3	99	63	01100011	c
52	34	00110100	4	100	64	01100100	d
53	35	00110101	5	101	65	01100101	e
54	36	00110110	6	102	66	01100110	f
55	37	00110111	7	103	67	01100111	g
56	38	00111000	8	104	68	01101000	h
57	39	00111001	9	105	69	01101001	i
58	3a	00111010	:	106	6a	01101010	j
59	3b	00111011	;	107	6b	01101011	k
60	3c	00111100	<	108	6c	01101100	l
61	3d	00111101	=	109	6d	01101101	m
62	3e	00111110	>	110	6e	01101110	n
63	3f	00111111	?	111	6f	01101111	o
64	40	01000000	@	112	70	01110000	p
65	41	01000001	A	113	71	01110001	q
66	42	01000010	B	114	72	01110010	r
67	43	01000011	C	115	73	01110011	s
68	44	01000100	D	116	74	01110100	t
69	45	01000101	E	117	75	01110101	u
70	46	01000110	F	118	76	01110110	v

71	47	01000111	G		119	77	01110111	w
72	48	01001000	H		120	78	01111000	x
73	49	01001001	I		121	79	01111001	y
74	4a	01001010	J		122	7a	01111010	z
75	4b	01001011	K		123	7b	01111011	{
76	4c	01001100	L		124	7c	01111100	|
77	4d	01001101	M		125	7d	01111101	}
78	4e	01001110	N		126	7e	01111110	~
79	4f	01001111	O		127	7f	01111111	DELETE

Example: checking if a character is a lowercase alphabet

```
1. int main()
2. {
3.     char c1;
4.     scanf("%c", &c1);
5.     /* the ascii code of c1 must lie between the ascii
   codes of ‘a’ and ‘z’ */
6.     if (c1 >= 'a' && c1<= 'z')
7.         printf("%c is a lowercase alphabet\n", c1);
8.     else
9.         printf("%c is not a lowercase alphabet\n", c1);
10.    return 0;
11.}
```

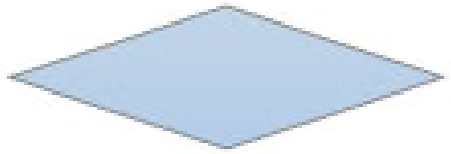
Example: converting a character from lowercase to uppercase

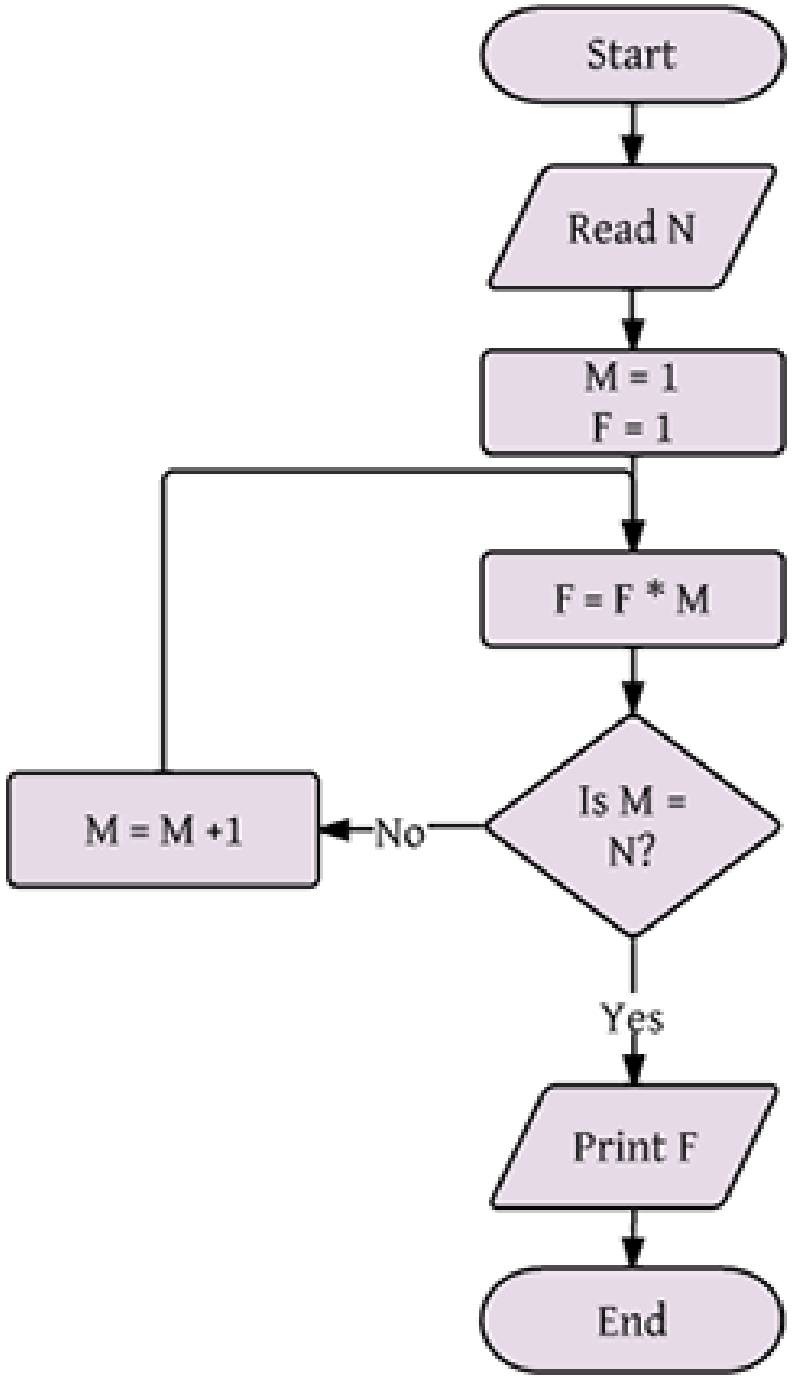
```
1.  #include<stdio.h>
2.  int main()
3.  {
4.      char c1;
5.      scanf("%c", &c1);
6.      /* convert to uppercase if lowercase, else leave as it is */
7.      if (c1 >= 'a' && c1<= 'z')
8.      /* since ascii codes of uppercase letters are contiguous, the
    uppercase version of c1 will be as far
9.      away from the ascii code of ‘A’ as it is from the ascii code of ‘a’ */
10.         c1 = 'A' + (c1 - 'a');
11.     printf("The letter is %c\n", c1);
12.     return 0;
13. }
```

Problem solving

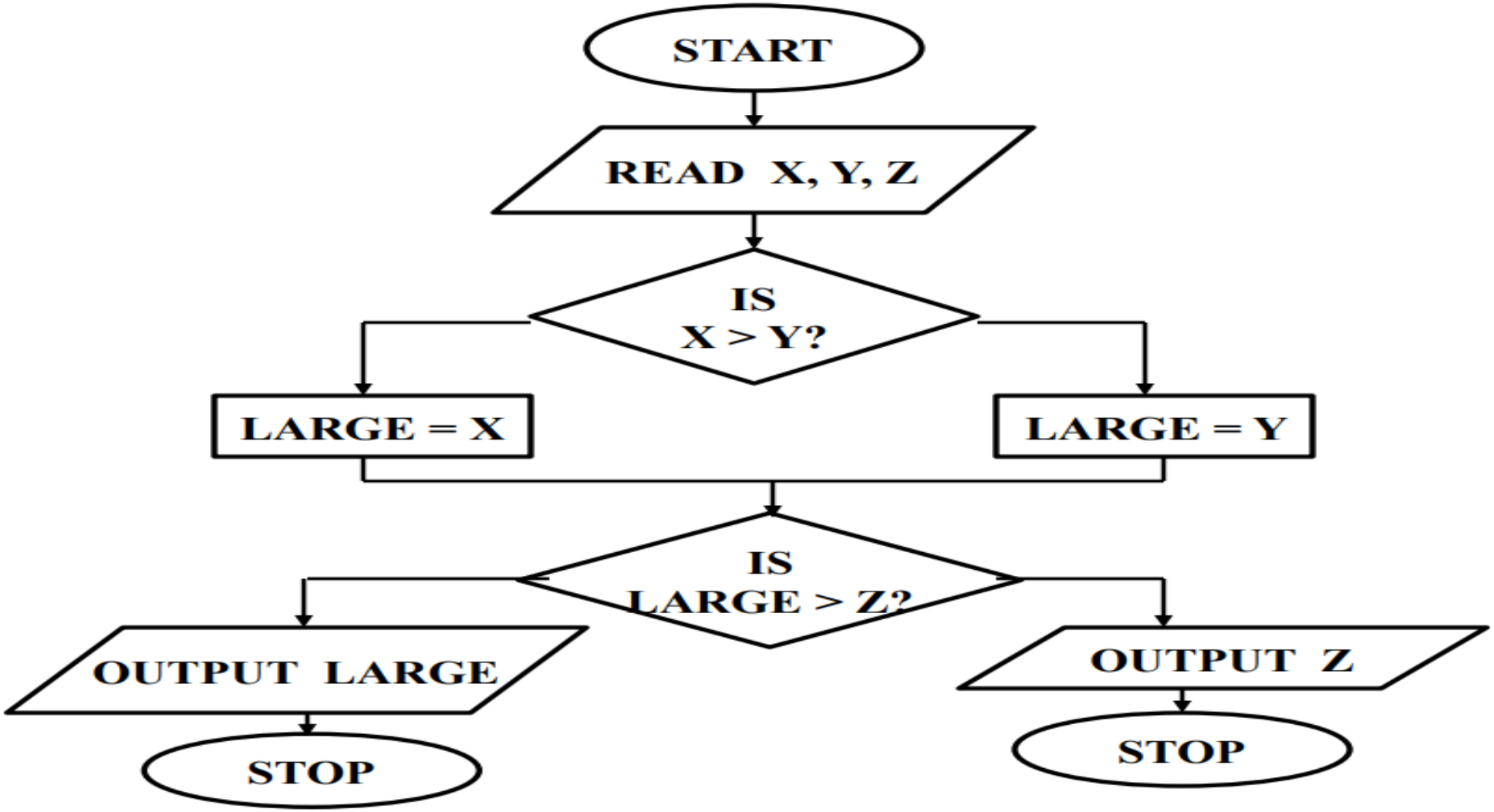
- Step 1:
  - Clearly specify the problem to be solved.
- Step 2:
  - Draw flowchart or write algorithm.
- Step 3:
  - Convert flowchart (algorithm) into program code.
- Step 4:
  - Compile the program into executable file.
- Step 5:
  - For any compila1on error, go back to step 3 for debugging.
- Step 6:
  - Execute the executable file (program).

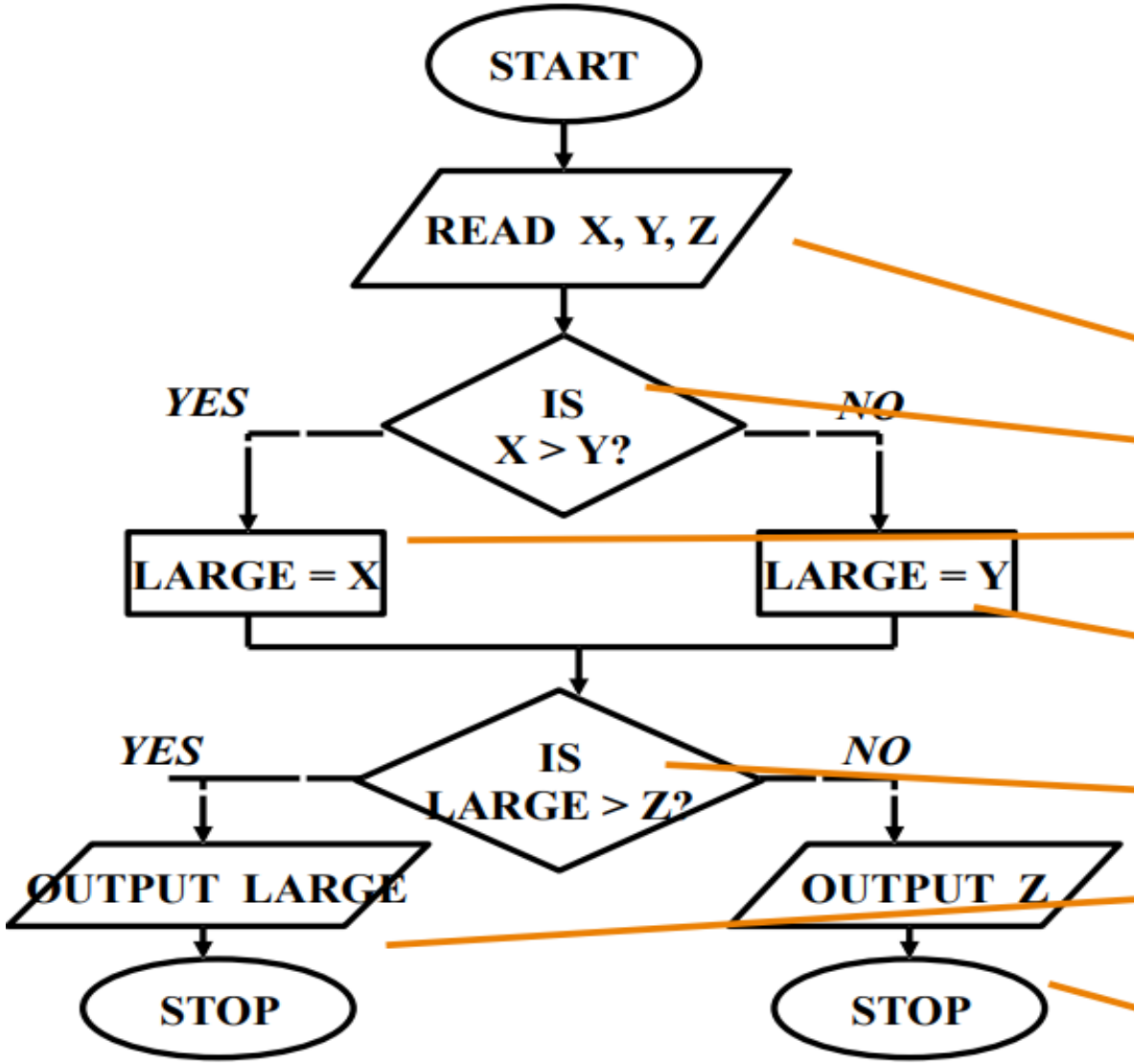
Flowchart – types of boxes

	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision


Start
Read N
M = 1
F = 1
F = F * M
Is M = N?
No
M = M +1
Yes
Print F
End

Example 2: find the largest among three numbers






```
#include <stdio.h>
int main()
{
    int  X,Y,Z,Large;
    scanf ("%d %d %d", &X, &Y, &Z);
    if (X>Y)     {
       Large=X;
     }     else {
    Large=Y;
   }
   if (Large>Z)
      printf ("\nThe largest number
  is: %d",Large);
     else
      printf ("\n The largest
  number is: %d",Z);
   return 0;
}
```

Desirable Programming Style

- **Clarity**
  - The program should be clearly written.
  - It should be easy to follow the program logic.
- **Meaningful variable names**
  - Make variable/constant names meaningful to enhance program clarity.
    - 'area' instead of 'a'
    - 'radius' instead of 'r'
- **Program documentation**
  - Insert comments in the program to make it easy to understand.
  - Never use too many comments.
- **Program indentation**
  - Use proper indentation.
  - Structure of the program should be immediately visible.

Bitwise Operator

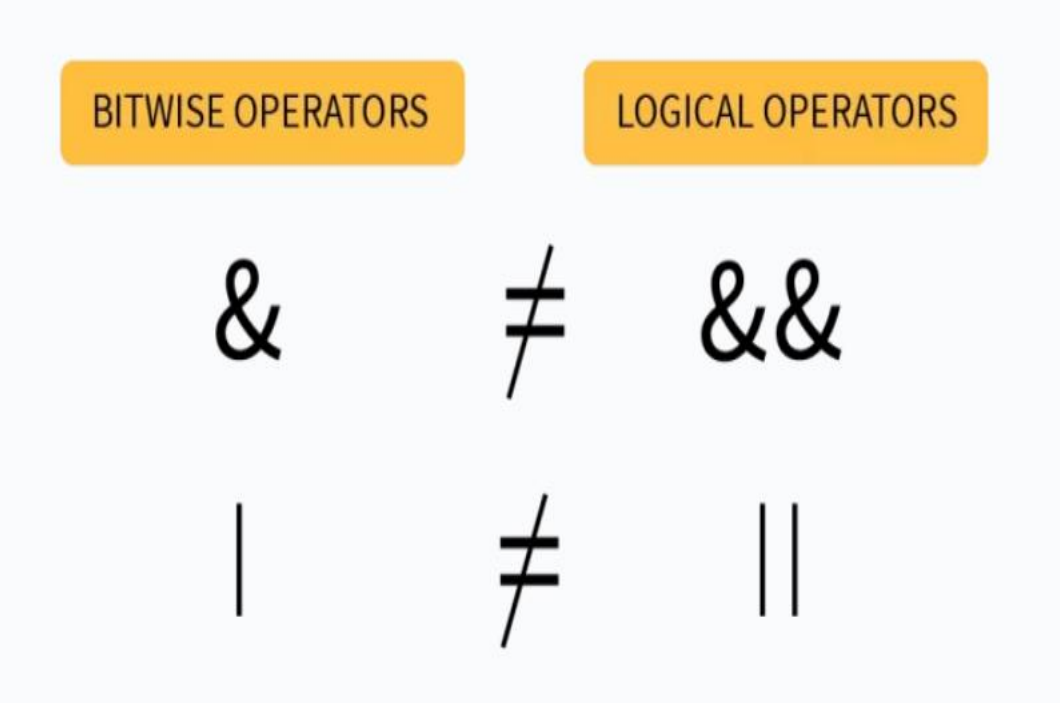
There are **6 bitwise operators** in C language. They are

- AND (&)
- OR (|)
- XOR (^)
- COMPLEMENT (~)
- Left Shift (<<)
- Right Shift (>>)

BITWISE operators

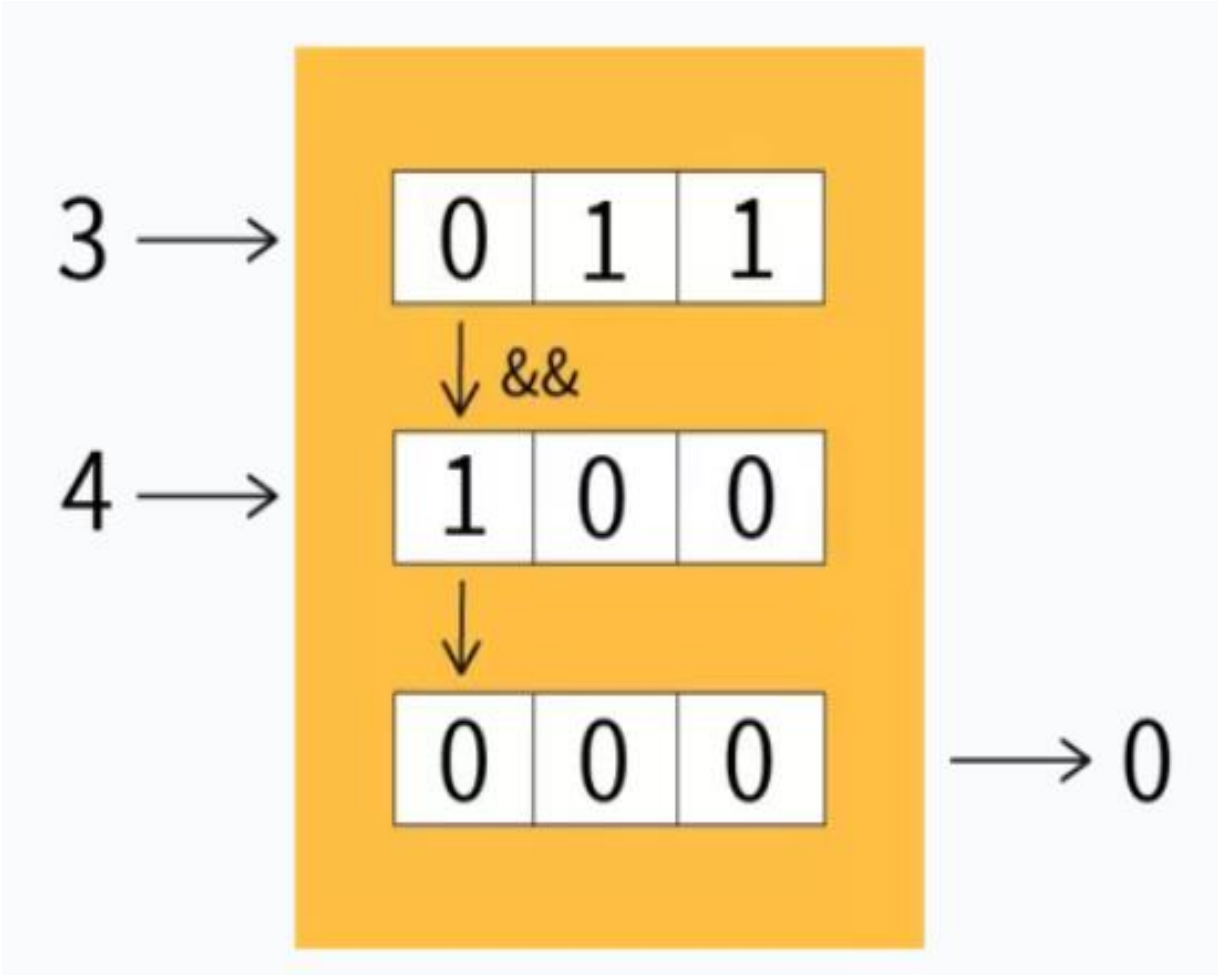
x	y	x&y	x|y	x^y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bitwise operators vs Logical operators in C



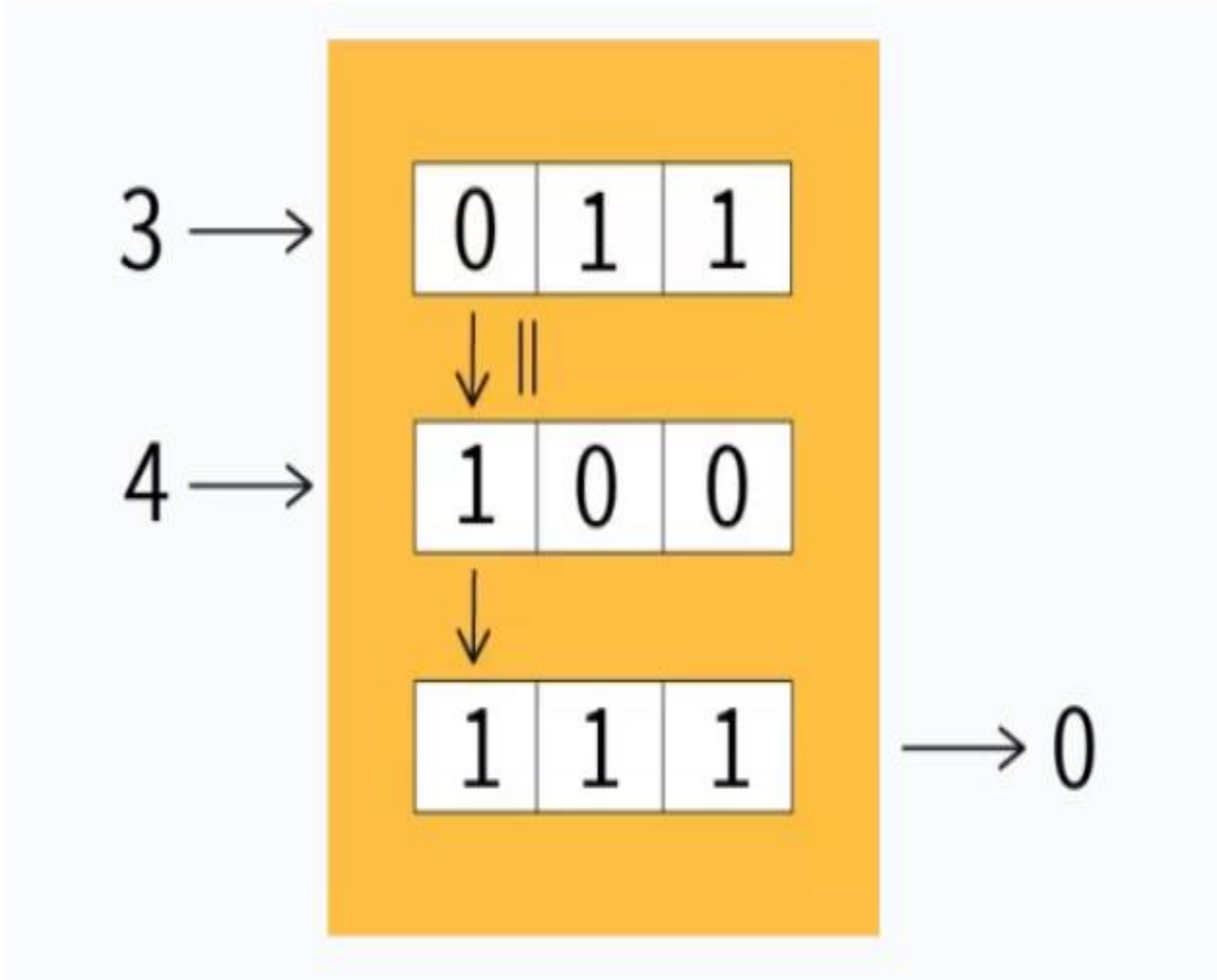
Bitwise &

- int ans, num1 = 3, num2 = 4;
- ans = num1 & num2;
- printf("3 & 4 = %d", ans);



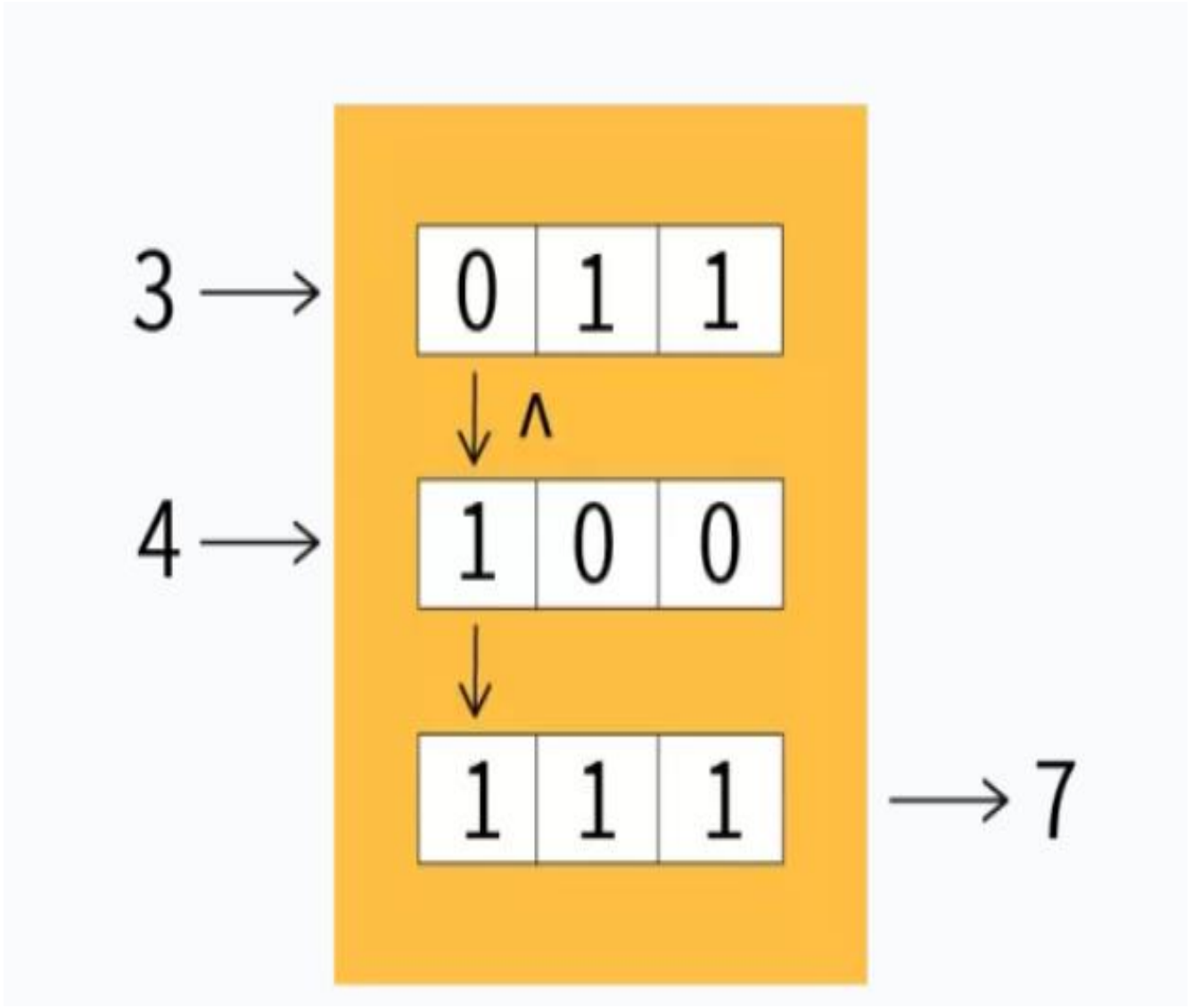
Bitwise |

- int ans, num1 = 3, num2 = 4;
- ans = num1 | num2;
- printf("3 | 4 = %d", ans);



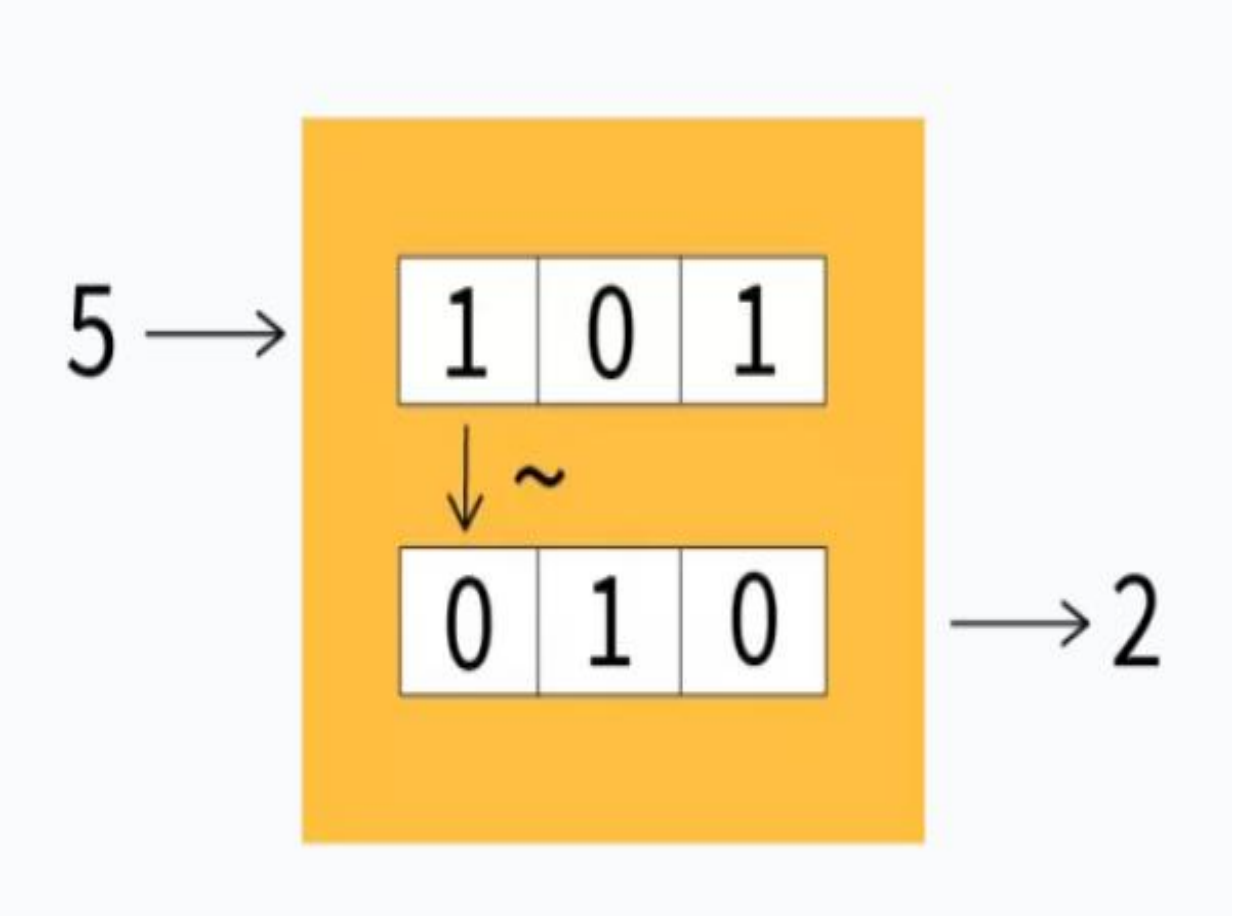
BITWISE XOR

- int ans, num1 = 3, num2 = 4;
- ans = num1 ^ num2;
- printf("3 ^ 4 = %d", ans);



BITWISE ~

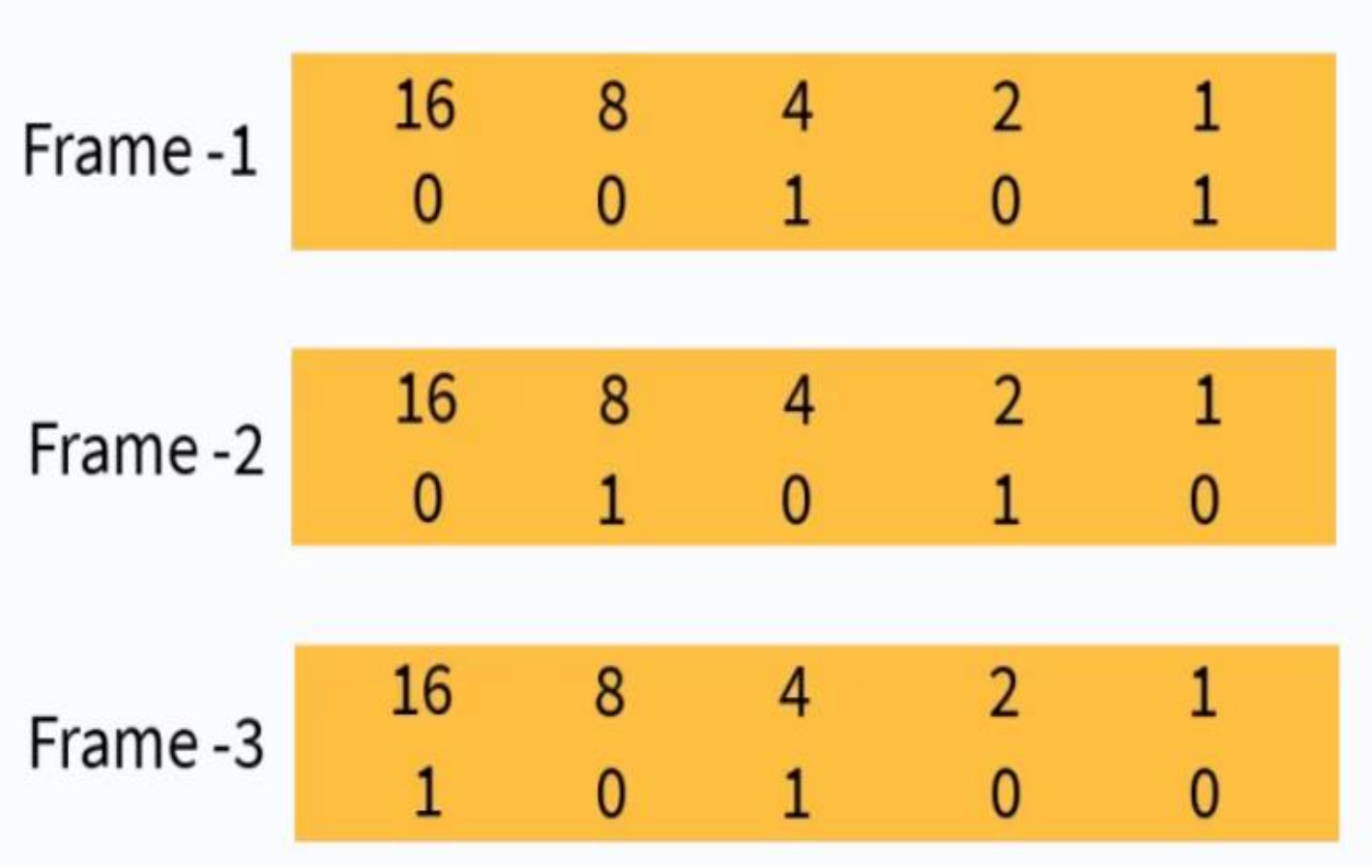
- int ans, num1 = 5;
- ans = ~num1;
- printf("~5 = %d", ans);



Shift Left <<

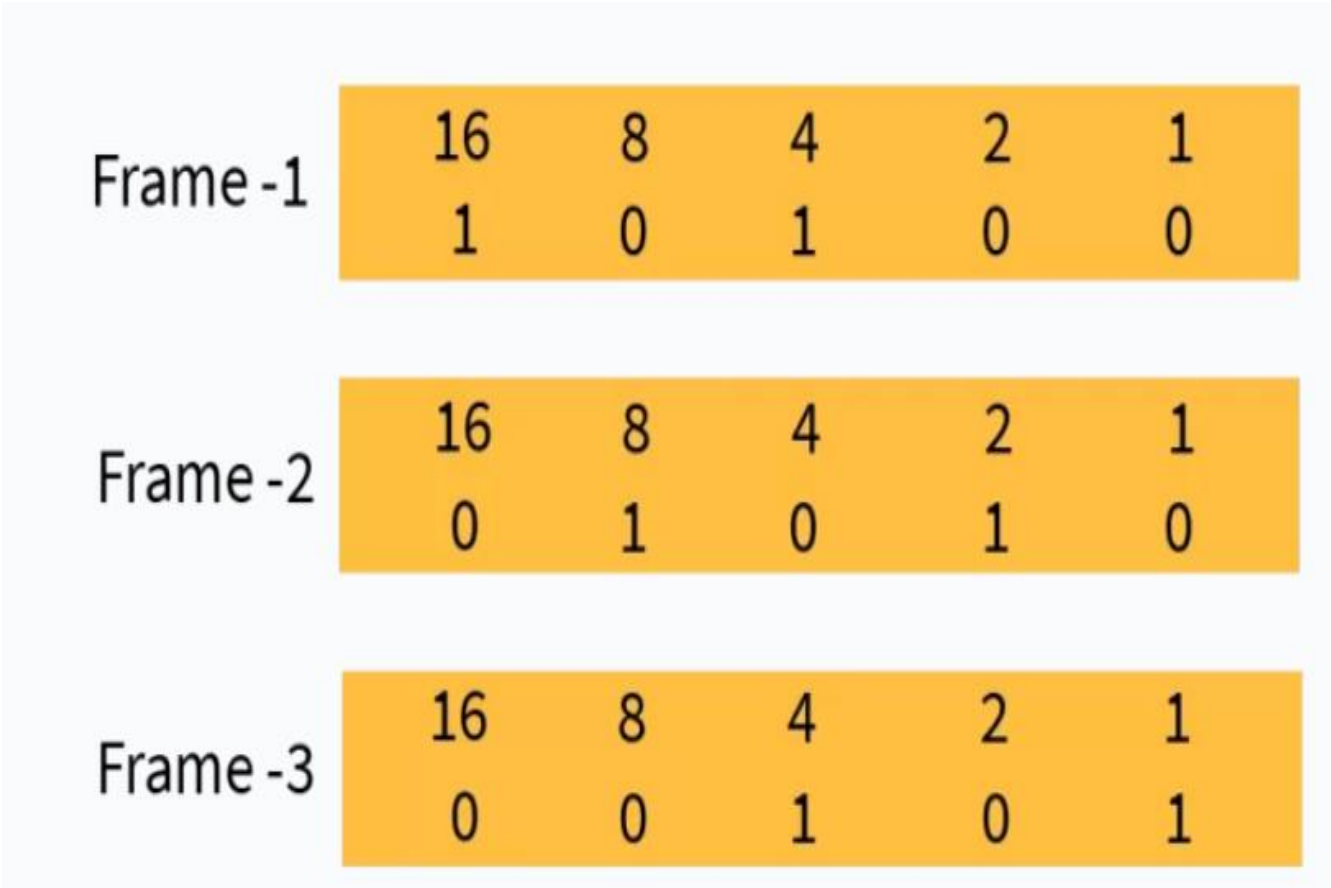
- The shift left operator shifts the bit pattern of an integer value by a specified number of bits to the left.

- int ans, num1 = 5;
- ans = num1 << 2;
- printf("5 << 2 = %d", ans);



Shift Right >>

- The shift right operator is almost similar to the shift left operator, the only difference is that it shifts the bit to the bits to right instead of left.
- int ans, num1 = 20;
- ans = num1 >> 2;
- printf("20 >> 2 = %d", ans);



sizeof() in C

- The sizeof() operator is used to find out the size of the variable, pointer, array, or expression passed to it.
- It is generally useful in finding out the amount of memory space in bytes needed to store a variable, pointer, array, or the result of an expression. eg6

Example 4

- WAP to check the age based quota in Indian Railway ticketing system.
- Kid Quota : 0-5
- Child Quota : upto 10
- Otherwise : General
- Above 65: Senior Quota
- Draw Flowchart and pseudocode

Indentation Example :: Good Style

```
/* A program to check the age based quota in Indian Railway ticketing system */

#include <stdio.h>
#define SENIOR      60          /* Declare the age of Senior Citizen */

int main()
{
        int age;
        scanf("%d",&age);
        if(age< SENIOR) {
                if(age<5) {
                        printf("Kid Quota");
                } else if (age<10) {
                        printf("Child Quota");
                } else {
                        printf("General Quota");
                }
        } else {
                printf("Senior Citizen");
        }
        return 0;
}
```

Example 3:
Grade computation

MARKS ≥ 90 → Ex

89 ≥ MARKS ≥ 80 → A

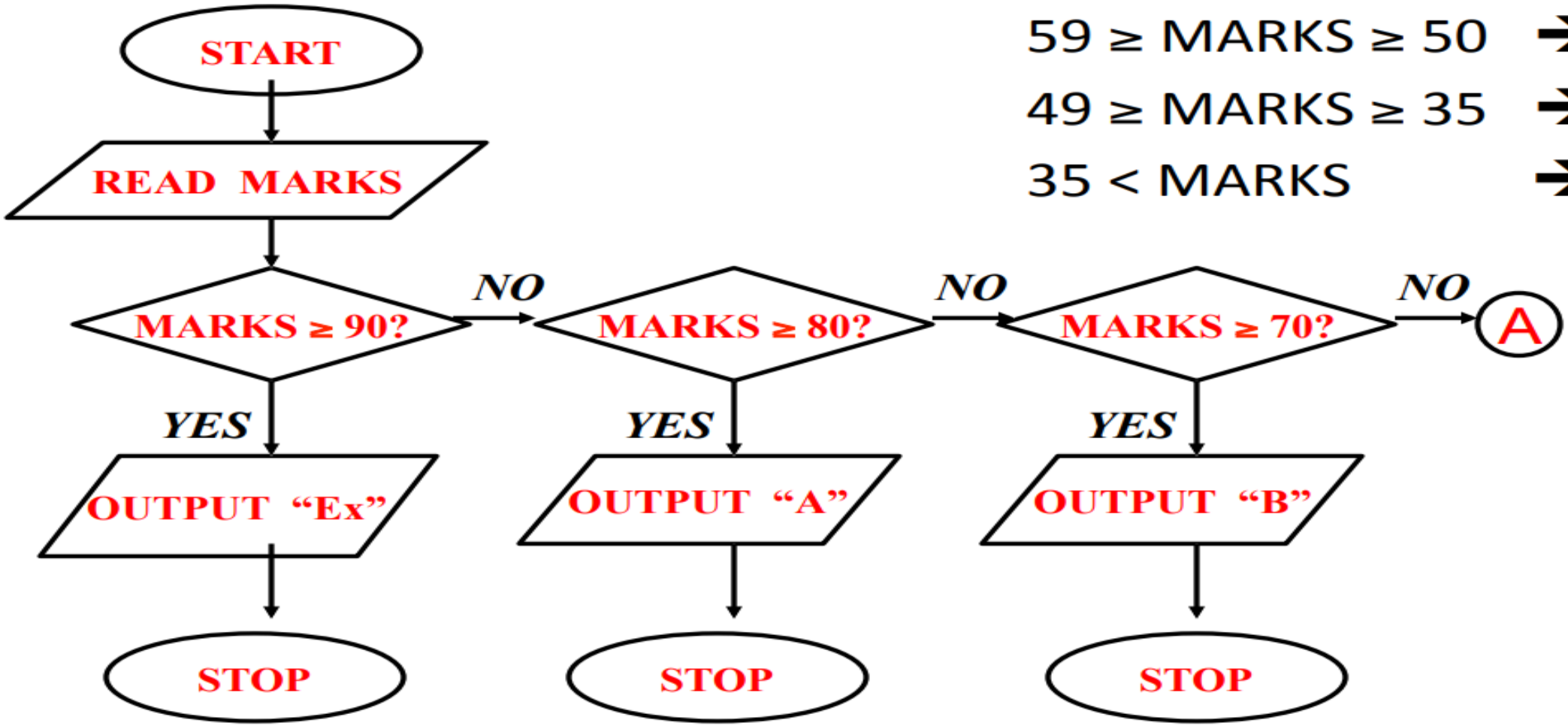
79 ≥ MARKS ≥ 70 → B

69 ≥ MARKS ≥ 60 → C

59 ≥ MARKS ≥ 50 → D

49 ≥ MARKS ≥ 35 → P

35 < MARKS → F



Ternary conditional operator (?:)

- Takes three arguments (condition, value if true, value if false).
- Returns the evaluated value accordingly.

```
(condition1)? (expr1): (expr2);
```

```
age >= 60 ? printf("Senior Citizen\n") : printf("General Quota\n")
```

Example:

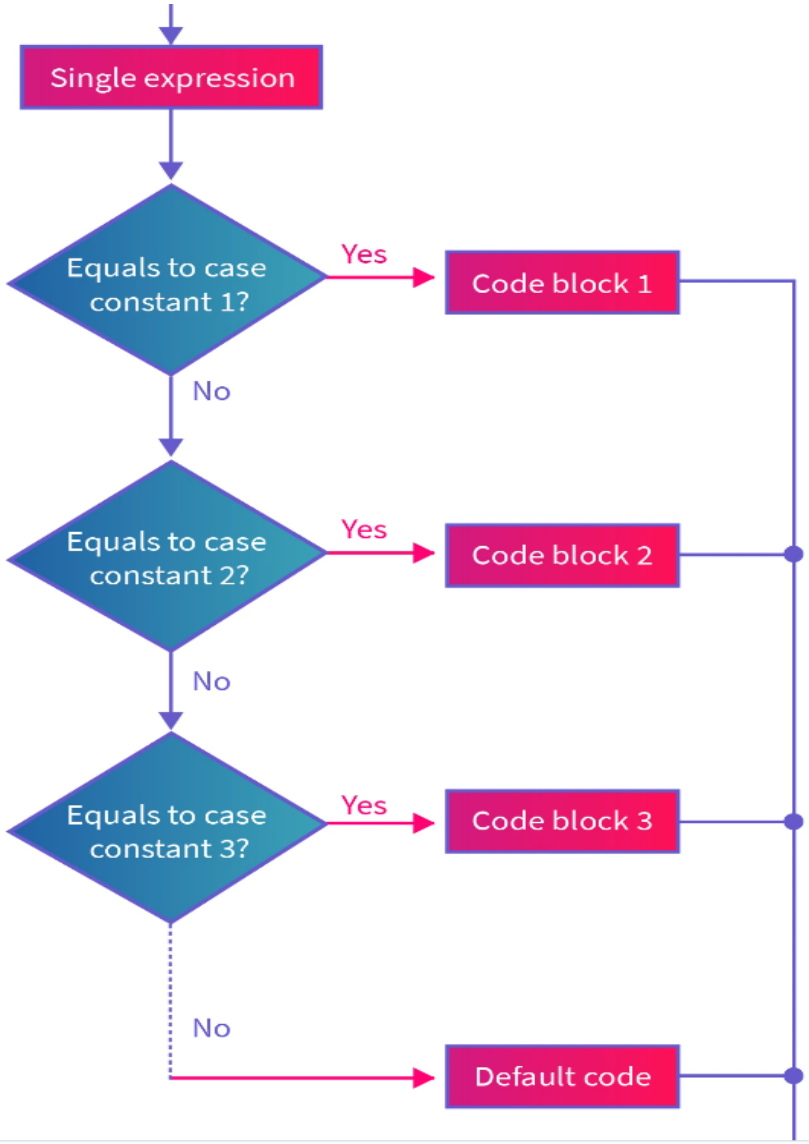
```
bonus = (basicPay<18000) ? basicPay*0.30 : basicPay*0.05;
```

Returns a value

Switch Case

Syntax of the Switch Statement in C Programming Language.

```
switch(expression)
{
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 //code to be executed if all cases are not matched;
}
```



switch example

```
switch ( letter ) {
   case 'A':
    printf("First letter\n");
    break;
   case 'Z':
    printf("Last letter\n");
    break;
   default :
    printf("Middle letter\n");
    break;
}
```

"break" statement is used to break the order of execution.

Behavior of switch

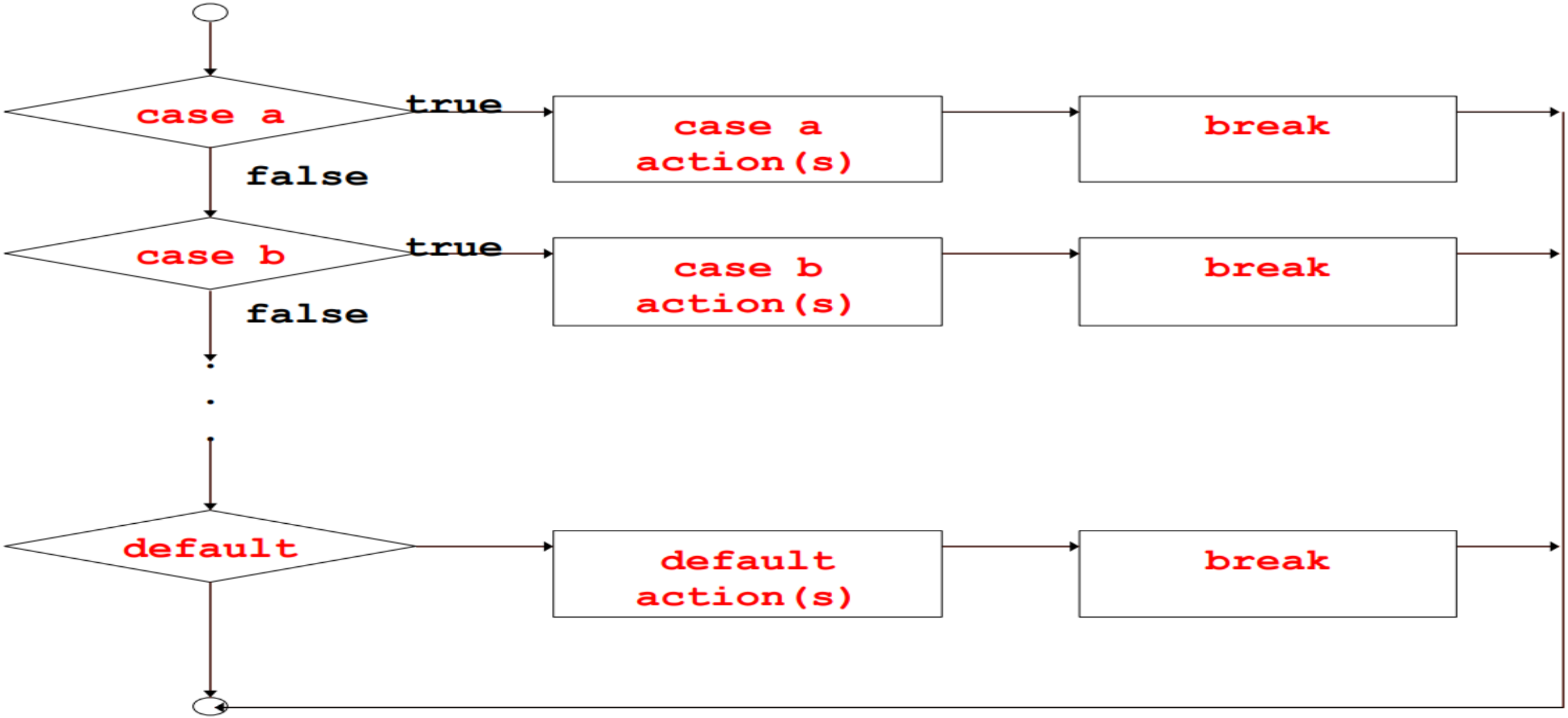
```
switch (expression)  {
        case const-expr-1: S-1
        case const-expr-2: S-2
      :
        case const-expr-m: S-m
        default: S
}
```

- expression is first evaluated
- It is then compared with const-expr-1, const-expr-2,…for equality in order
- If it matches any one, all statements from that point till the end of the switch are executed (including statements for default, if present)
  - Use break statements if you do not want this (see example)
- Statements corresponding to default, if present, are executed if no other expression matches

The break Statement

- Used to exit from a switch or terminate from a loop.
- With respect to “switch”, the “break” statement causes a transfer of control out of the entre “switch” statement, to the first statement following the “switch” statement

Flowchart for switch statement



Example: switch break

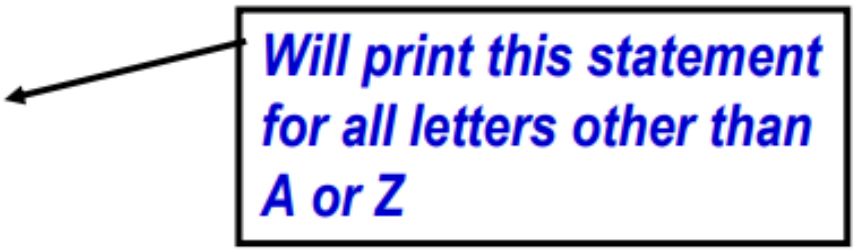
```
switch   (primaryColor = getchar())   {

   case 'R': printf ("RED \n");
        break;

   case 'G': printf ("GREEN \n");
        break;

   case 'B': printf ("BLUE \n");
        break;
   default:  printf ("Invalid Color \n");
        break;          /* break is not mandatory here
   */
}
```

Switch statement

```
main(){
int a=10;
switch(a) {
 case 20: printf("a=20\n"); break;
 case 10: printf("a=10\n"); break;
 default: printf("unknown");
}}  //output is a=10
```

Example

```
switch ( letter ) {
        case 'A':
                printf ("First letter \n");
                break;
        case 'Z':
                printf ("Last letter \n");
                break;
        default :
                printf ("Middle letter \n");
                break;
}
```



Example

```
switch ( choice = getchar( ) ) {
    case ‘r’ :
    case ‘R’: printf(“Red”);
              break;
    case ‘b’ :
    case ‘B’ : printf(“Blue”);
               break;
    case ‘g’ :
    case ‘G’: printf(“Green”);
               break;
    default: printf(“Black”);
}
```

Since there isn’t a break *statement here, the control passes to the next statement* (printf) *without checking the next condition.*

Another way

```
switch  ( choice = toupper( getchar( ) ) )  {
            case 'R': printf ("RED \n");
                                break;
            case 'G': printf ("GREEN \n");
                                break;
            case 'B': printf ("BLUE \n");
                                break;
            default:   printf ("Invalid choice \n");
}
```

Rules for Switch Statement in C Language

- The expression provided in the switch statement should always be either an integer value or a char value.

```
switch(x==5) // this is valid.
switch(x/3) // this is not valid.
```

- You can't use two case labels with the same value. Duplicate case values would throw an error.

```
switch(x)
{
    case a:
        // set of statement
    case a:
        //set of statements
}// this kind of switch statement will throw you a compile-time
```

- You can't define ranges within a case statement, nor can you use a single case label for more than one value.

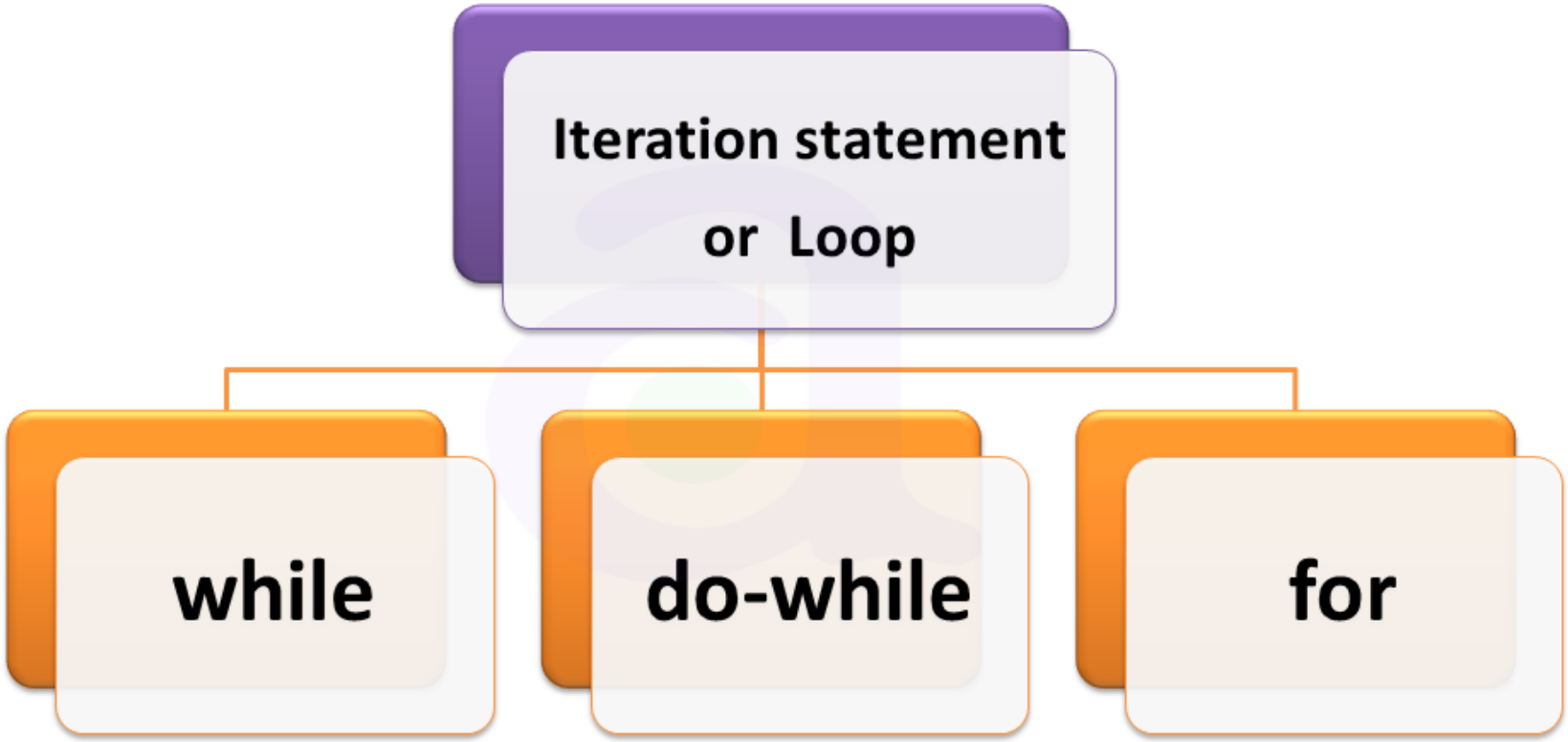
```
switch(x)
{
    case 1,10:
        // set of statements
    case 2-5:
        //set of statements
}// these 2 case statements will also throw a compile time error
```

Switch	If Else
The Expression is tested for equality only .	The expression cam be tested for inequality as well. (<,>)
Only one value is used to match against all case labels.	Multiple expression can be tested for branching.
Switch case is not effective when checking for a range value.	If else is a better option to check ranges.
Switch case cannot handle floating point values	If else can handle floating point values
The case label must be constant(Characters of integers).	If else can use variables also for conditions.
Switch statement provides a better way to check a value against a number of constants .	If else is not suitable for this porpose .

Practice Problems

- Read in 3 integers and print a message if any one of them is equal to the sum of the other two.
- Read in the coordinates of two points and print the equation of the line joining them in y = mx +c form.
- Read in the coordinates of 3 points in 2-d plane and check if they are collinear. Print a suitable message.
- Read in the coordinates of a point, and the center and radius of a circle. Check and print if the point is inside or outside the circle.
- Read in the coefficients a, b, c of the quadratic equation ax2 + bx + c = 0, and print its roots nicely (for imaginary roots, print in x + iy form)
- Suppose the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are mapped to the lowercase letters a, b, c, d, e, f, g, h, i, j respectively. Read in a single digit integer as a character (using %c in scanf) and print its corresponding lowercase letter. Do this both using switch and without using switch (two programs). Do not use any ascii code value directly.
- Suppose that you have to print the grades of a student, with >= 90 marks getting EX, 80-89 getting A, 70-79 getting B, 60-69 getting C, 50-59 getting D, 35-49 getting P and <30 getting F. Read in the marks of a student and print his/her grade.

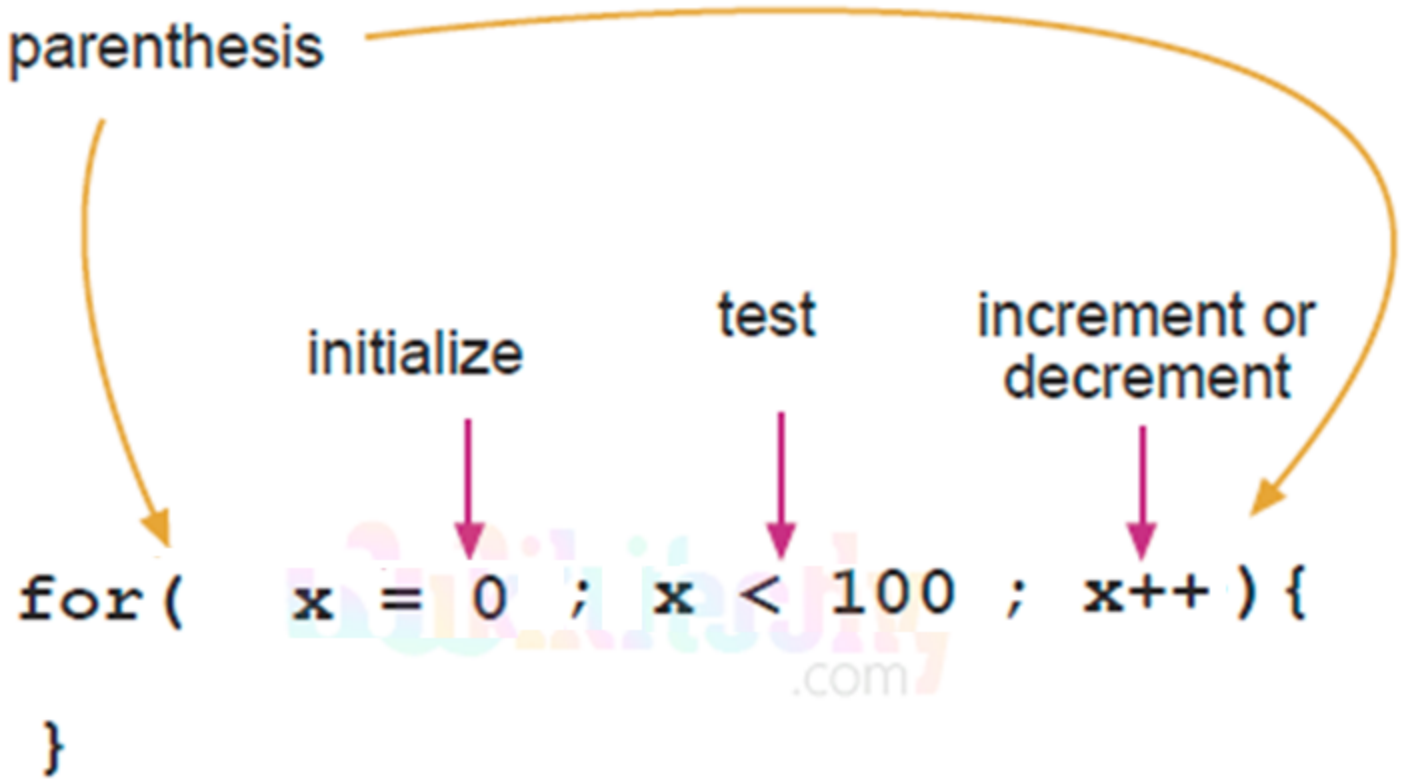
Iterative statements - *repetitions*

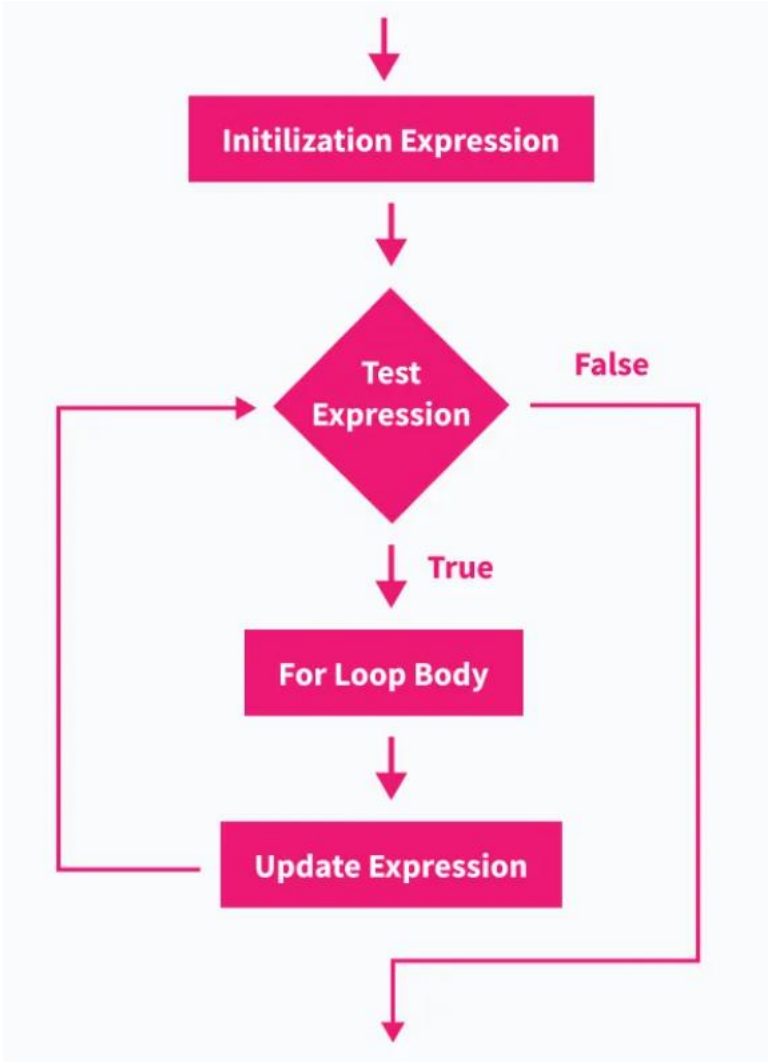


The Essentials of Repetition

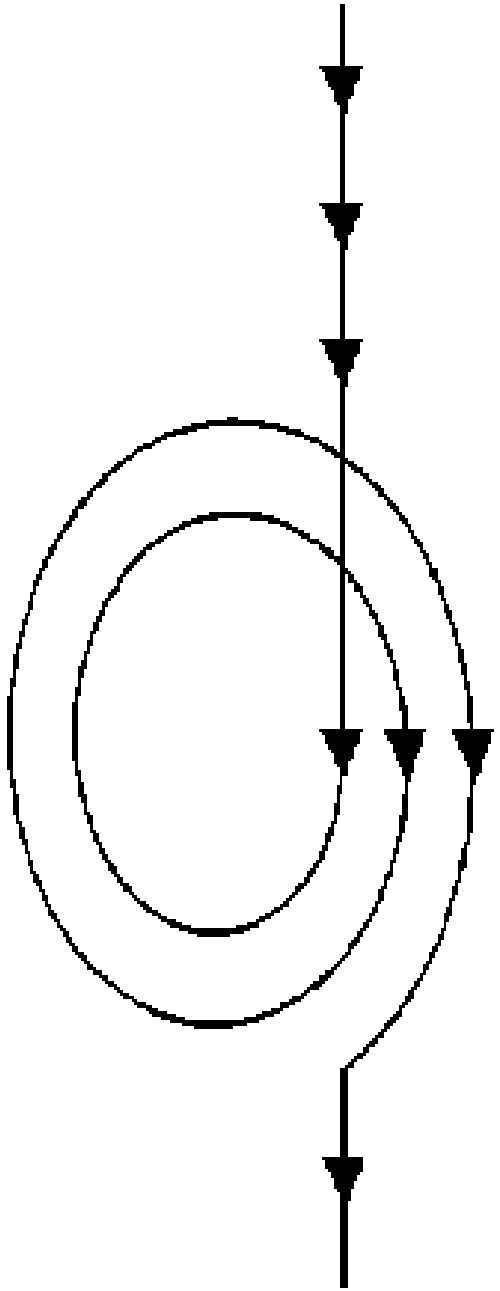
- Loop
  - Group of instruc1ons computer executes repeatedly while some condition remains true
- Counter-controlled repetition
  - Definite repetition
  - know how many times loop will execute
  - Control variable used to count repetition
- Sentinel-controlled repetition
  - Indefinite repetition
  - Used when number of repetitions not known
  - Sentinel value indicates "end of data"

A **for loop** is a **control structure** that is used to run a block of instructions multiple times until a certain condition fails.





for loop



```
int data1 = 100;

int i;

data1 =  data1 + 10;


for (i = 0; i < 3; i++)
{
    data1 = data1 + 20;

}

data1 = data1 + 30;
```

Forms of for loop

A) NO INITIALIZATION:

Initialization can be skipped as shown below:

```
int x = 10;
for( ; x < 50; x++)
```

B) NO UPDATION:

We can run a for loop without needing an updation in the following v

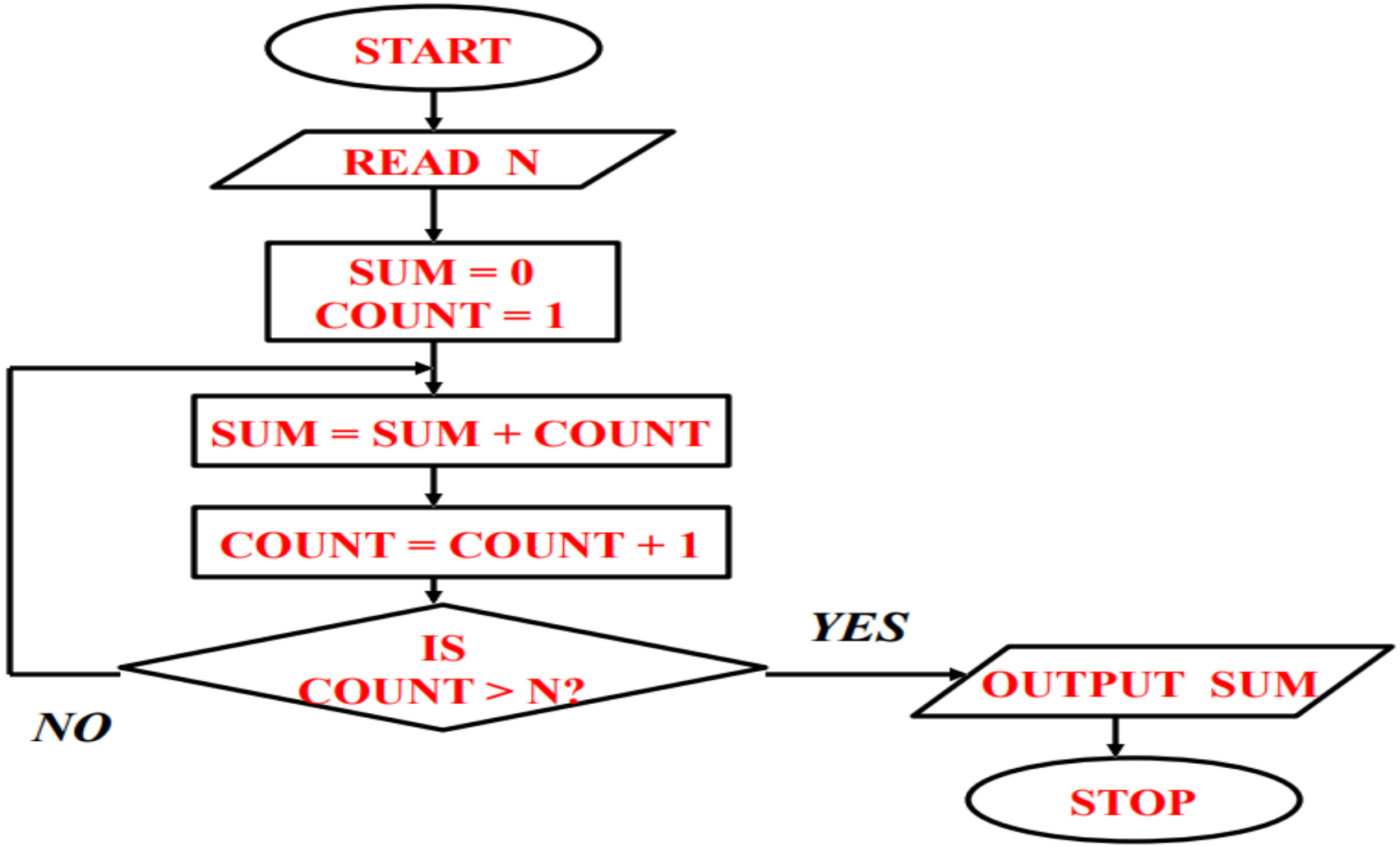
```
int num;
for (num = 50; num < 60; ) {
    num++;
}
```

C) NO INITIALIZATION AND UPDATE STATEMENT:

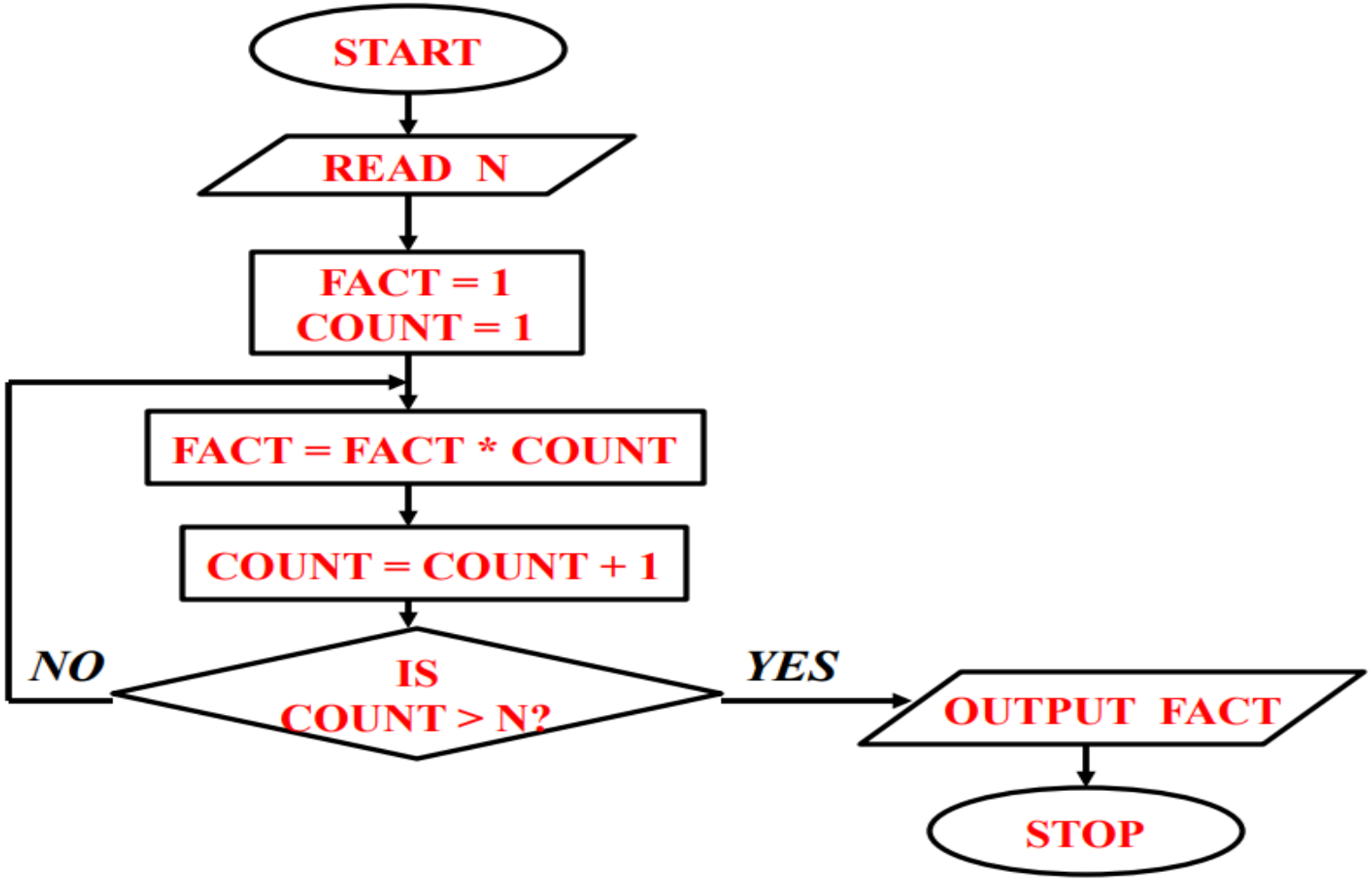
We can avoid both initialization and update statements too!

```
int x = 20;
for (; x < 40; ) {
    x++;
}
```

Example 6: Sum of first N natural numbers



Example 7: Computing Factorial



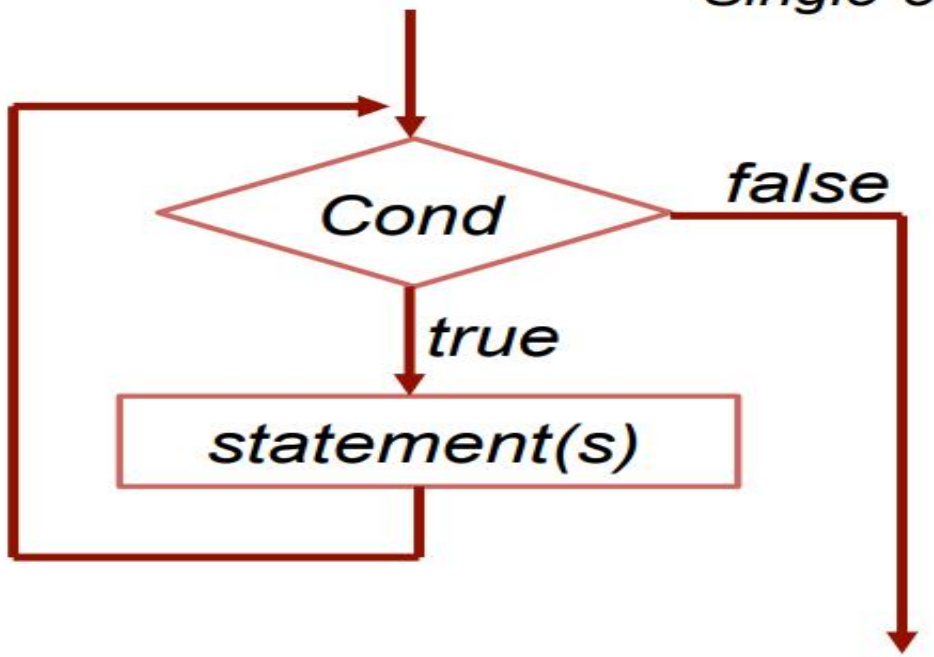
Counter-Controlled Repetition

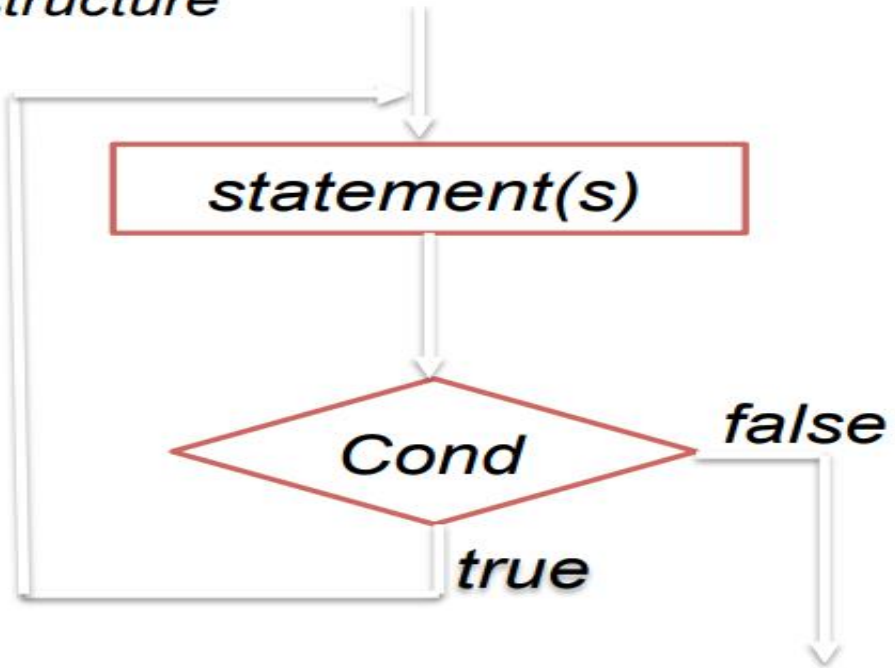
- Counter-controlled repetition requires
  - *name* of a control variable (or loop counter).
  - *initial value* of the control variable.
  - condition that tests for the *final value* of the control variable (i.e., whether looping should continue).
  - *increment* (or *decrement*) by which the control variable is modified each time through the loop.

```
int counter =1;                          /* initialization */
while (counter <= 10) {    /* repetition condition */
      printf( "%d\n", counter );
      ++counter;                              //increment
}
```

Repetition: Flowchart

Single-entry / single-exit structure





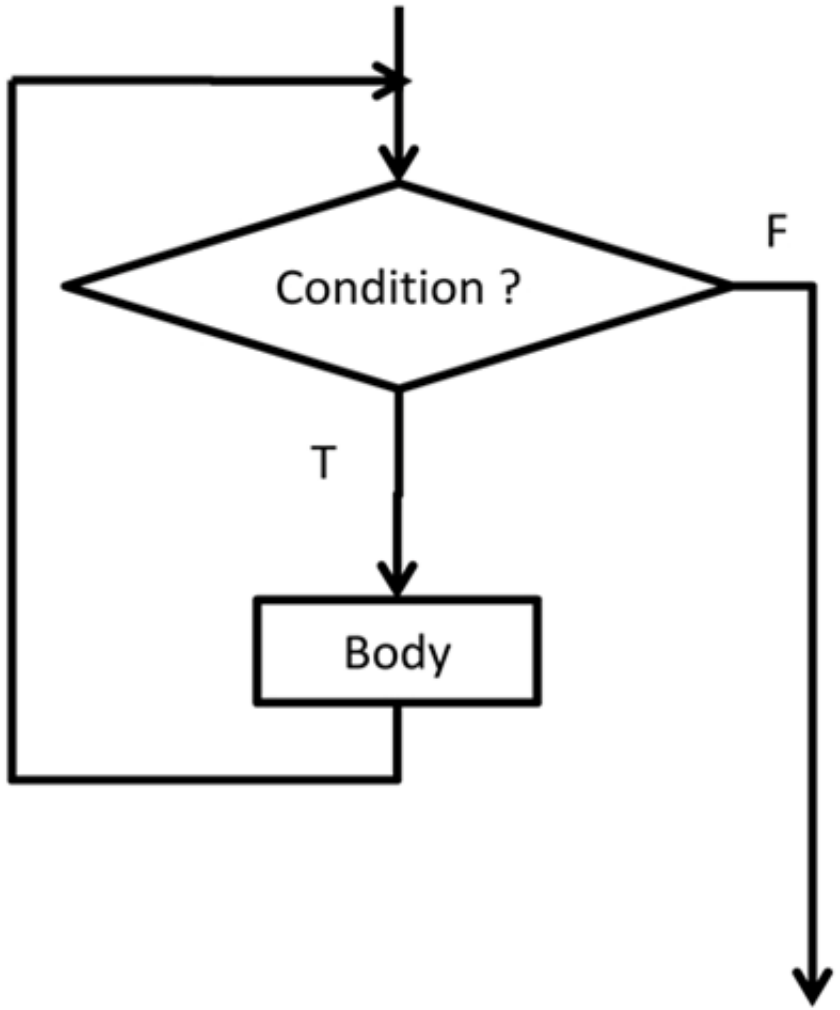

```
int counter =1;

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

May not execute at all based on condition.

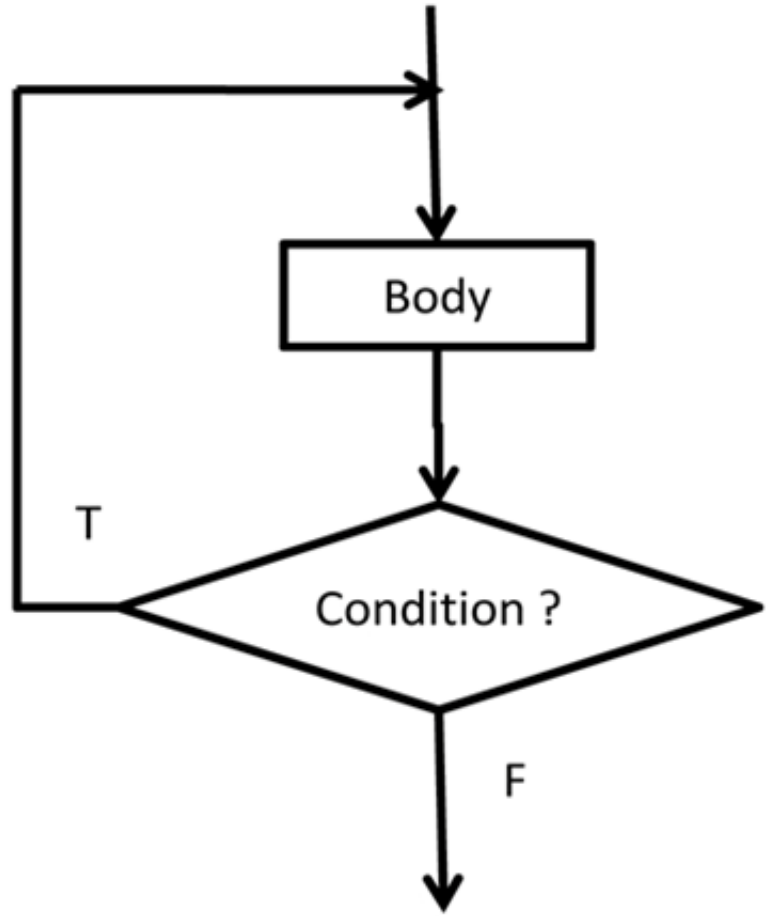
```
int counter =1;

do {
        printf( "%d\n", counter );
        ++counter;
} while (counter <= 10) ;
```

Will be executed at least once, whatever be the condition.

```
while( condition )
    body;
```




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



while, do-while Statement

```
while (condition)
   statement_to_repeat;

while (condition) {
    statement_1;
    ...
    statement_N;
}
```

```
int digit = 0;
while (digit <= 9)
 printf ("%d \n", digit++);
```

```
int weight=75;
while ( weight > 65 ) {
    printf("Go, exercise, ");
    printf("then come back. \n");
    printf("Enter your weight: ");
    scanf("%d", &weight);
  }
```

```
weight=75;
do {
    printf("Go, exercise, ");
    printf("then come back. \n");
    printf("Enter your weight: ");
    scanf("%d", &weight);
  } while ( weight > 65 ) ;
```

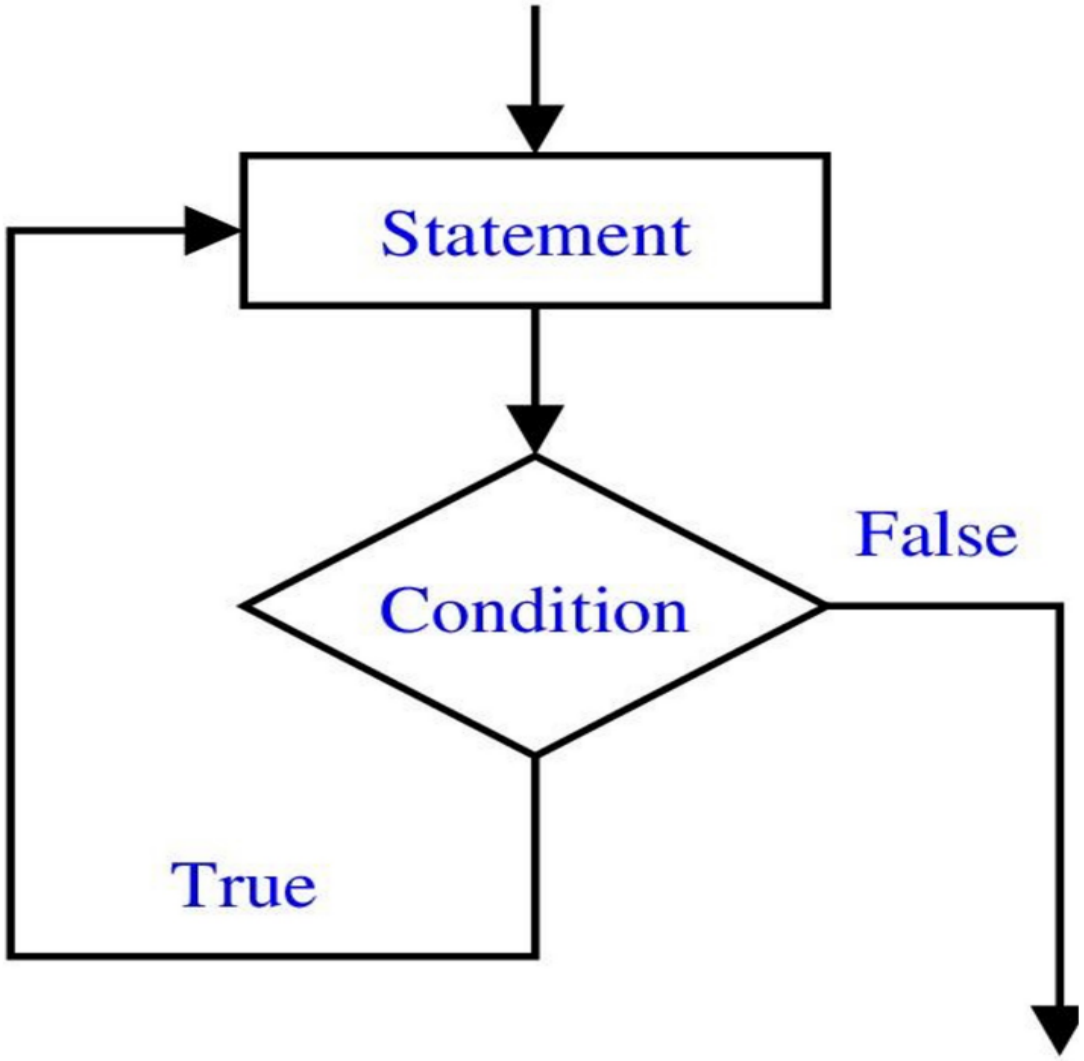
```
do {
    statement-1;
    statement-2;
        ……..
     statement-n;
} while ( condition ) ;
```

At least one round of exercise is ensured.

do-while statement

```
do
    statement;
while (expression);

do {
    Block of statements;
} while (expression);
```

```
main () {
      int digit=0;
      do
         printf("%d\n",digit++);
      while (digit <= 9) ;
}
```



```
int main()
{
        int i=1, n;
        scanf("%d", &n);
        while (i <= n)  {
            printf ("Line no : %d\n",i);
            i = i + 1;
        }
}
```

Output

```
4
Line no : 1
Line no : 2
Line no : 3
Line no : 4
```

Factorial using while

```
main(){
int i=1,n=5,f=1;
while(i<=n){
    f = f*i;
    i++;
}
    printf("%d",f);
}
```

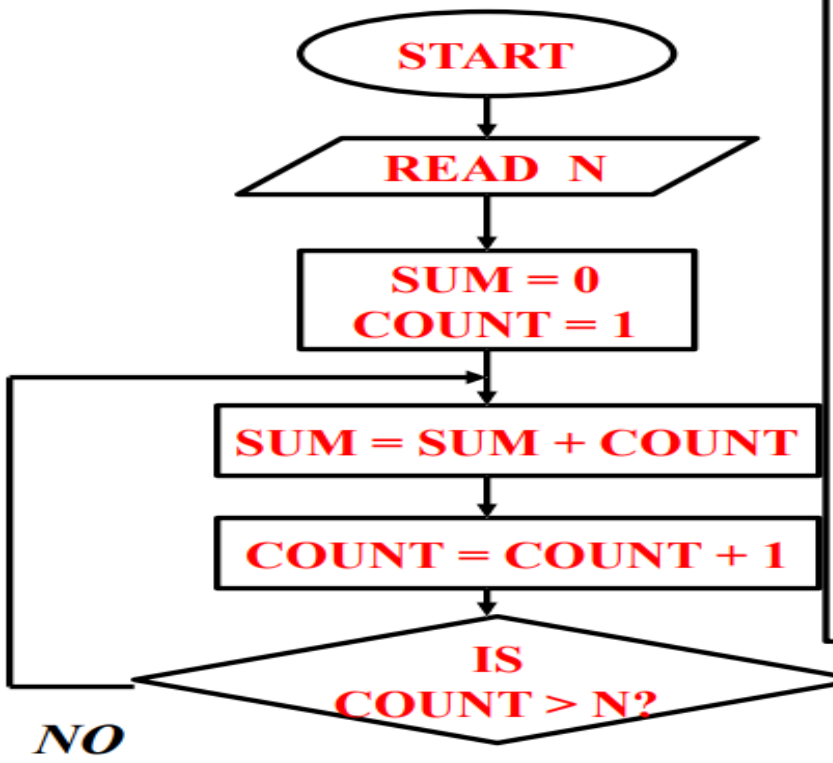
Factorial using while

```
main(){
int i=1,n=5,f=1;
do{
    f = f*i;
    i++;
} while(i<=n);
    printf("%d",f);
}
```

Factorial using for

```
main(){
int i,n=5,f=1;
for(i=1;i<=n;i++){
    f = f*i;
}
    printf("%d",f);
}
```

Sum of first N natural numbers




```
#include <stdio.h>

int main()
{
        int n, sum, count;
        printf("Enter a natural number: ");
        scanf("%d",&n);
        count=0;
        sum=0;
        while(count<=n) {
                sum+=count;   // sum=sum+count
                count++;
        }
        printf("The sum of first %d natural numbers is: %d",
                        n,sum);
        return 0;
}
```

Line break in a statement is allowed after a comma.

Double your money

Suppose your Rs 10000 is earning interest at 1% per month. How many months until you double your money?

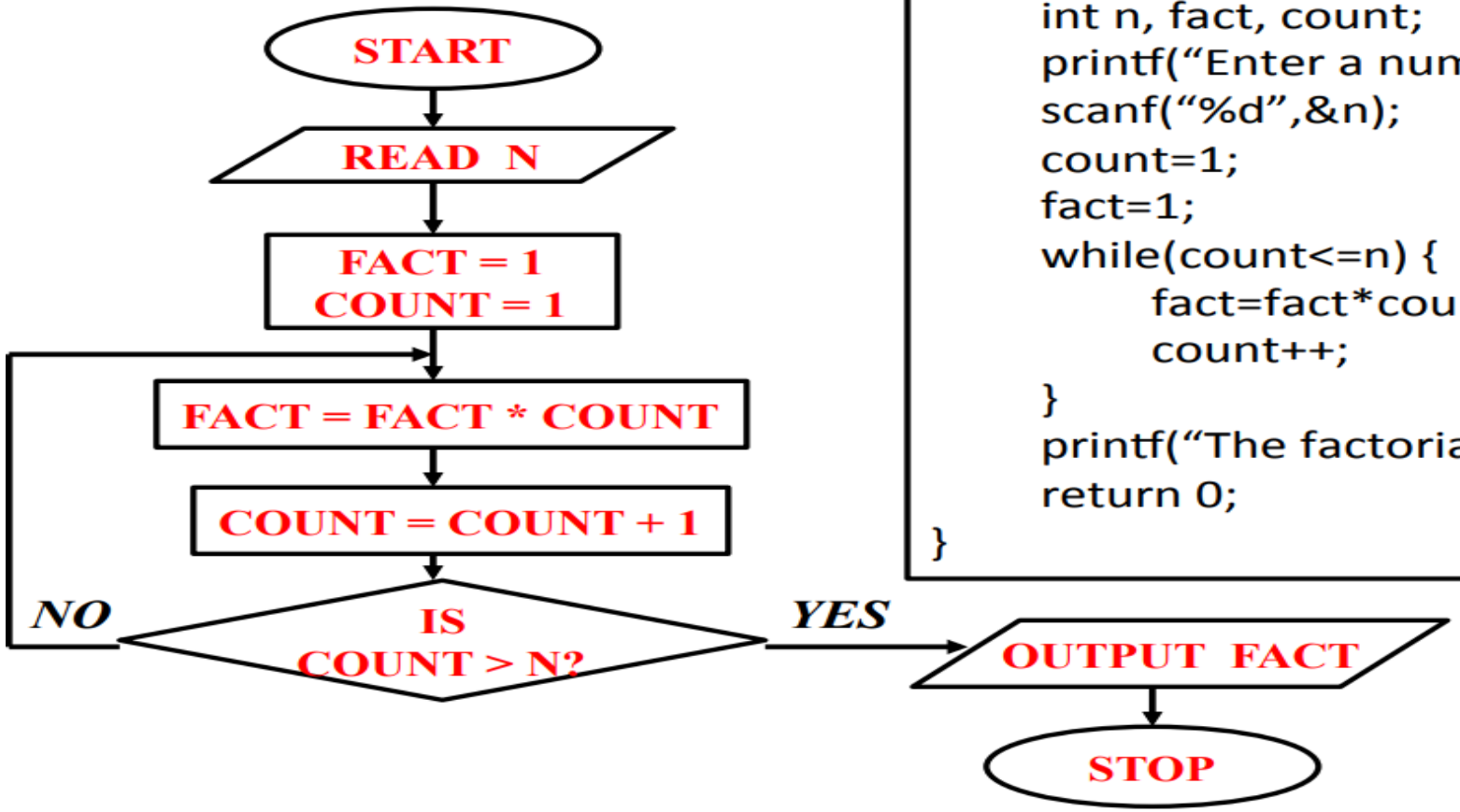
```
int main() {
    double my_money = 10000.0;
    int n=0;
    while (my_money < 20000.0) {
        my_money = my_money * 1.01;
        n++;
    }
  printf ("My money will double in %d months.", n);
  return 0;
}
```

Maximum of inputs

```
int main() {

    double max = 0.0, next;

    printf ("Enter positive numbers only, end with 0 or a negative number\n");

    scanf("%lf", &next);

    while (next > 0)  {

         if (next > max)  max = next;

         scanf("%lf", &next);

    }

    printf ("The maximum number is %lf\n", max) ;

    return 0;

}
```

```
Enter positive numbers only, end with 0 or a negative number
45
32
7
5
0
The maximum number is 45.000000
```

Computing Factorial




```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        count=1;
        fact=1;
        while(count<=n) {
                fact=fact*count;
                count++;
        }
        printf("The factorial of %d is: %d",n,fact);
        return 0;
}
```

Considering large factorial value, you may declare *fact* as float.

Which expressions decide how long a while loop will run?

```
int main()
{
    int i = 1, n;
    scanf("%d", &n);
    while (i <= n)  {
        printf ("Line no : %d\n",i);
        i = i + 1;
    }
}
```

Initialization of the loop control variable

Test condition

Update of the loop control variable in each iteration

Next, we will see another way of writing the same loop, where these 3 parts will be written together

Equivalence of for and while

```
for ( expr1; expr2; expr3)
        statement;
```

Same as

```
expr1;
while (expr2)  {
      statement
      expr3;
}
```

Computing Factorial

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        count=1;
        fact=1;
        while(count<=n) {
                fact=fact*count;
        count++;
        }
        printf("%d",fact);
        return 0;
}
```

count may increment.

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        count=n;
        fact=1;
        while(count>=1) {
                fact=fact*count;
        count--;
        }
        printf("%d",fact);
        return 0;
}
```

count may decrement.

Loop variable may decrement

Computing Factorial

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        count=1;
        fact=1;
        while(count<=n) {
                fact=fact*count;
        count++;
        }
        printf("%d",fact);
        return 0;
}
```

while loop

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        fact=1;
        for(count=1;count<=n;count++) {
                fact=fact*count;
        }
        printf("%d",fact);
        return 0;
}
```

for loop

Computing Factorial

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        count=1;
        fact=1;
        for(;count<=n;) {
                fact=fact*count;
        count++;
        }
        printf("%d",fact);
        return 0;
}
```

for loop working as while

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        fact=1;
        for(count=1;count<=N;count++) {
                fact=fact*count;
        }
        printf("%d",fact);
        return 0;
}
```

for loop

Homework:
Rewrite the factorial using *for* loop and by decrementing count.

Computing Factorial

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        fact=1;
        for(count=1;count<=n;count++) {
                fact=fact*count;
        }
        printf("%d",fact);
        return 0;
}
```

for loop

```
#include <stdio.h>

int main()
{
        int n, fact, count;
        printf("Enter a number: ");
        scanf("%d",&n);
        for(fact=1,count=1;count<=n;count++) {
                fact=fact*count;
        }
        printf("%d",fact);
        return 0;
}
```

for loop with comma operator

The comma operator:

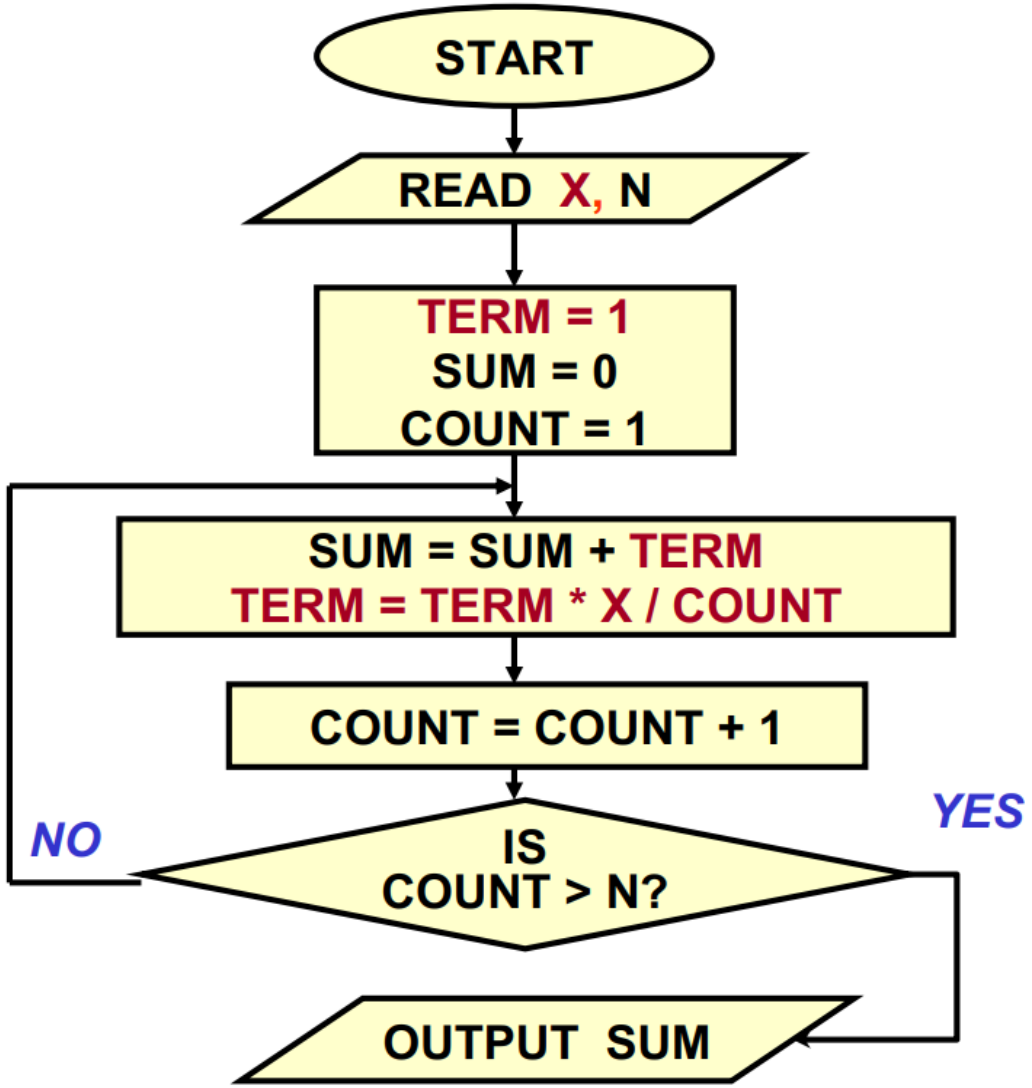
We can give several statements separated by commas in place of "expression1", "expression2", and "expression3".

Sum of first N natural numbers

```
int main () {
      int N, count, sum;
      scanf ("%d", &N) ;

      sum = 0;
      count = 1;
      while (count <= N)   {
            sum = sum + count;
            count++;
      }

      printf ("Sum = %d\n", sum) ;
      return 0;
}
```

```
int main () {
      int N, count, sum;
      scanf ("%d", &N) ;

      sum = 0;
      for (count=1; count <= N; count++)
            sum = sum + count;

      printf ("Sum = %d\n", sum) ;
      return 0;
}
```

Example: *Computing e^x series up to N terms (1 + x + (x^2 / 2!) + (x^3 / 3!) + …)*




```
int main () {
     float x, term, sum;
     int n, count;
     scanf ("%f", &x);
     scanf ("%d", &n);
     term = 1.0; sum = 0;
     for (count = 1; count <= n; ++count)   {
          sum += term;
          term *= x/count;
      }
     printf ("The series sum is %f\n", sum);
     return 0;
}
```

Output

```
2.3
10
The series sum is 7.506626
```

- Problem: Prompt user to input “month” value, keep prompting until a correct value of month is given as input

```
do {
    printf (“Please input month {1-12}”);
     scanf (“%d”, &month);
} while ((month < 1) || (month > 12));
```

Advanced expression in for structure

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.
  - e.g. Let `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 )
```

"Increment" may be negative (decrement)

If loop continuation condition initially false

Body of *for* structure not performed
Control proceeds with statement after *for* structure

Specifying “Infinite Loop”

```
count=1;
while(1) {
        printf(“Count=%d”,count);
        count++;
}
```

```
count=1;
do {
        printf(“Count=%d”,count);
        count++;
} while(1);
```

```
count=1;
for(;;) {
        printf(“Count=%d”,count);
        count++;
}
```

```
for(count=1;;count++) {
        printf(“Count=%d”,count);
}
```

Specifying “Infinite Loop”

```
while   (1)   {
    statements
}
```

```
for   (; ;)
{
      statements
}
```

```
do   {
    statements
}   while (1);
```

break Statement

- Break out of the loop { }
  - can use with
    - *while*
    - *do while*
    - *for*
    - *switch*
  - does not work with
    - *if {}*
    - *else {}*
- Causes immediate exit from a while, for, do/while or switch structure
- Program execution continues with the first statement after the structure
- Common uses of the break statement
  - Escape early from a loop
  - Skip the remainder of a switch structure

Break from “Infinite Loop”

```
count=1;
while(1) {
        printf(“Count=%d”,count);
        count++;
        if(count>100)
                break;
}
```

```
count=1;
do {
        printf(“Count=%d”,count);
        count++;
        if(count>100)
                break;
} while(1);
```

```
count=1;
for(;;) {
        printf(“Count=%d”,count);
        count++;
        if(count>100)
                break;
}
```

```
for(count=1;;count++) {
        printf(“Count=%d”,count);
        if(count>100)
                break;
}
```

continue Statement

- continue
  - Skips the remaining statements in the body of a while, for or do/while structure
    - Proceeds with the next iteration of the loop
  - while and do/while
    - Loop-continuation test is evaluated immediately after the `continue` statement is executed
  - for structure
    - Increment expression is executed, then the loop-continuation test is evaluated.
    - *expression3* is evaluated, then *expression2* is evaluated.

An Example with break and continue

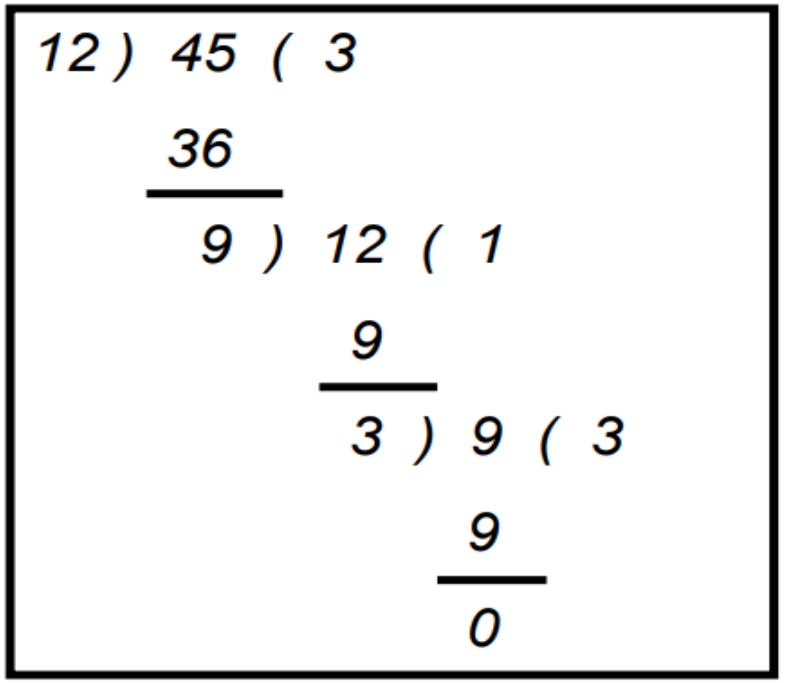
```
fact = 1;          /* a program to calculate 10 ! */
i = 1;
while  (1)  {
   fact = fact * i;
   i++ ;
   if(i<10) {
    continue;    /* not done yet! Go to next
   iteration*/
   }
   break;
}
```

Primality testing

```c
#include <stdio.h>
int main()
{
    int n, i=2;
    scanf ("%d", &n);
    while (i < n) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n);
            break;
        }
        i++;
    }
    if(i>=n)
        printf ("%d is a prime \n", n);
    return 0;
}
```

Compute GCD of two numbers

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int A, B, temp;
5.      scanf ("%d %d", &A, &B);
6.      if (A > B) {
7.          temp = A;
8.          A = B;
9.          B = temp;
10.     }
11.     while ((B % A) != 0) {
12.         temp = B % A;
13.         B = A;
14.         A = temp;
15.     }
16.     printf ("The GCD is %d", A);
17.     return 0;
18. }
```

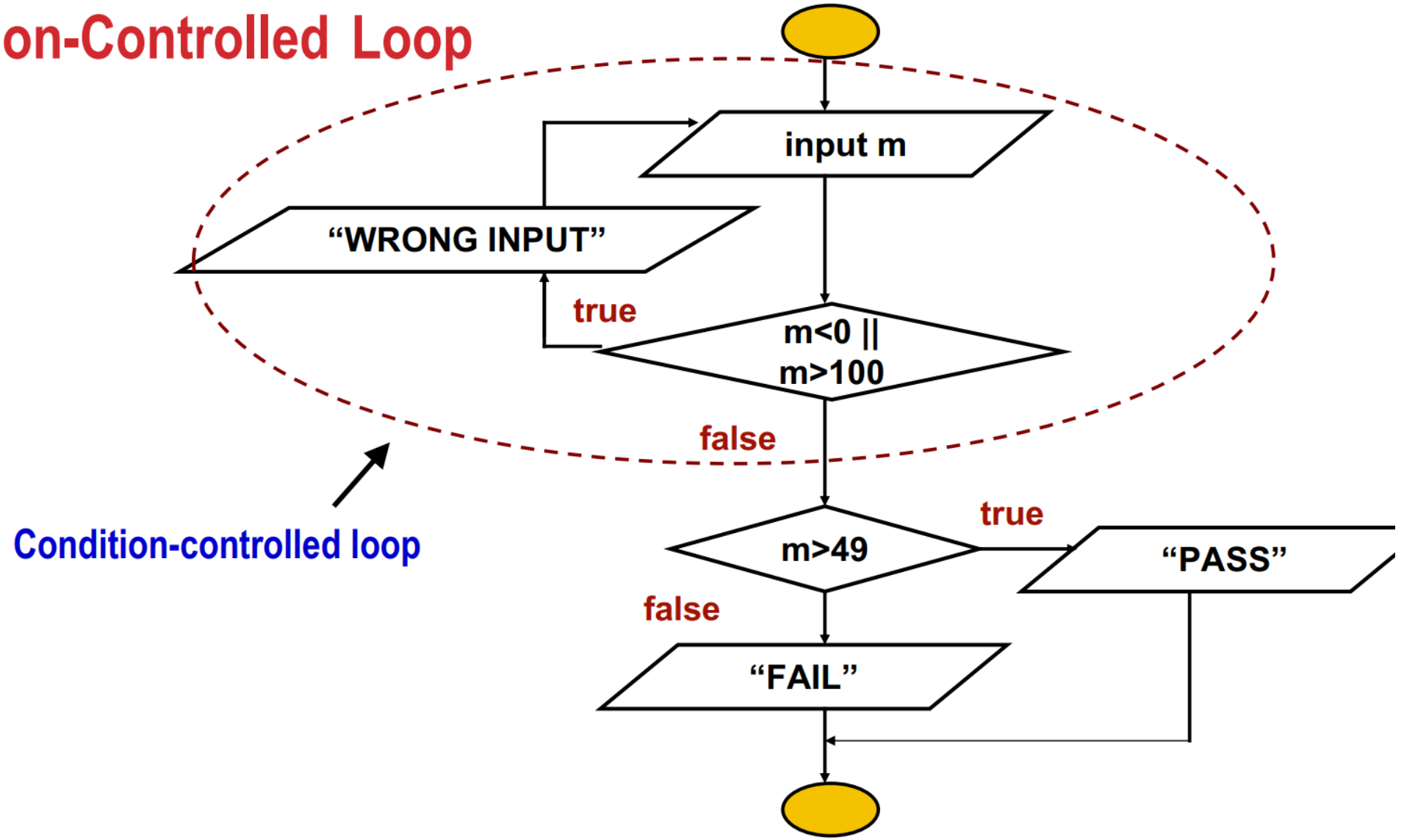


Initial: A=12, B=45
Iteration 1: temp=9, B=12,A=9
Iteration 2: temp=3, B=9, A=3
B % A = 0 ➔ GCD is 3

Grade: pass or Fail

- Given an exam marks as input, display the appropriate message based on the rules below:
- If marks is greater than 49, display “PASS”, otherwise display “FAIL”
- However, for input outside the 0-100 range, display “WRONG INPUT” and prompt the user to input again until a valid input is entered

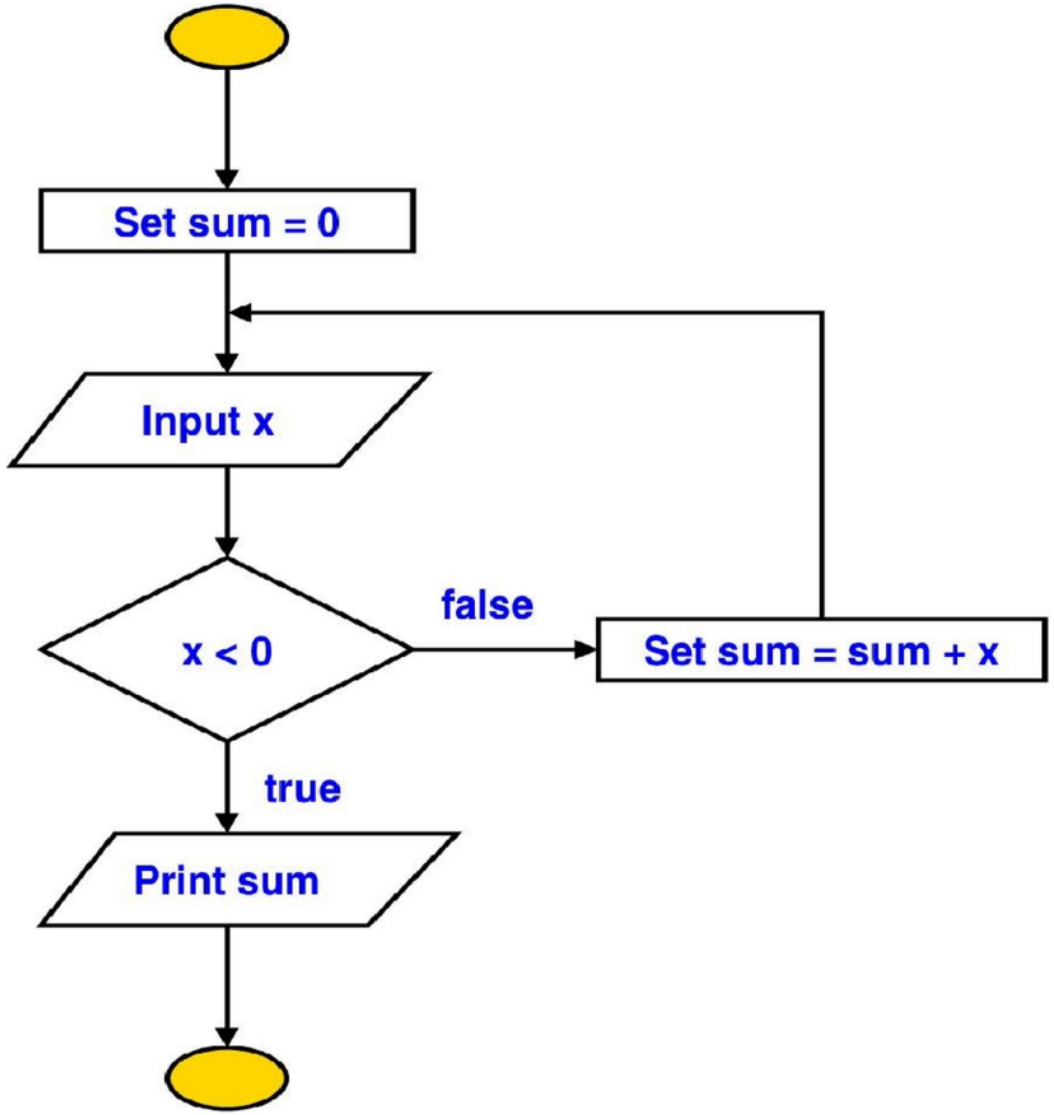
Condition-Controlled Loop



Input-Controlled Loop

Read a sequence of non-negative integers from the user, and display the sum of these integers.

A negative input indicates the end of input process.



Check Prime Number?

```c
#include<stdio.h>
int main() {
    int n, i=2;
    double limit;
    scanf ("%d", &n);
    limit = sqrt(n);
    for (i = 2, i <= limit; i++) {
        if (n % i == 0) {
            printf ("%d is not a prime \n", n);
            break;
        }
    }
    if (i > limit)
        printf ("%d is a prime \n", n);
    return 0;
}
```

Another Way

```
1.  #include<stdio.h>
2.  int main() {
3.      int n, i = 2, found = 0;
4.      double limit;
5.      scanf ("%d", &n);
6.      limit = sqrt(n);
7.      while (i <= limit) {
8.          if (n % i == 0) {
9.              found = 1; break;
10.         }
11.         i = i + 1;
12.     }
13.     if (found == 0)
14.         printf ("%d is a prime \n", n);
15.     else
16.         printf ("%d is not a prime \n", n);
17.     return 0;
18. }
```

Example with break and continue:

Add positive numbers until a 0 is typed, but ignore any negative numbers typed

```
1.  #include<stdio.h>
2.  int main() {
3.      int sum = 0, next;
4.      while (1) {
5.          scanf("%d", &next);
6.          if (next < 0)
7.              continue;
8.          if (next == 0)
9.              break;
10.         sum = sum + next;
11.     }
12.     printf ("Sum = %d\n", sum) ;
13.     return 0;
14. }
```

Some Loop Pitfalls

```
while (sum <= NUM) ;
        sum = sum+2;
```

```
for (i=0; i<=NUM; ++i);
        sum = sum+i;
```

```
for (i=1; i!=10; i=i+2)
        sum = sum+i;
```

Patterns!

- Helps
  - Loops
  - Algorithms
  - Using for and while loops
- the outer loop controls the number of rows,
- but the inner loop is used for displaying values in those columns.

Nested Loops: Printing a 2-D Figure

- How would you print the following diagram?

*****

*****

*****

repeat 3 times

print a row of 5 * s

Nested Loops (full program on next slide)

```
#define ROWS 3
#define COLS 5
...
row=1;
while (row <= ROWS) {
      /* print a row of 5 *'s */
       …
      printf("\n");
      row++;
}
```

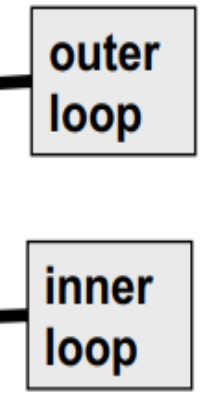
```
col=1;
while (col <= COLS) {
    printf ("* ");
    col++;
}
```

Nested Loops

For every iteration of the outer loop, the inner loop runs its entire length

```
const int ROWS = 3;
const int COLS = 5;
...
row = 1;
while (row <= ROWS) {
      /* print a row of 5 *'s */
        ...
      ++row;
}
```

```
row = 1;
while (row <= ROWS) {
      /* print a row of 5 *'s */
      col = 1;
      while (col <= COLS) {
            printf ("* ");
            col++;
      }
      printf("\n");
      ++row;
}
```



2-D Figure: with for loop

*****

*****

*****

```
const int ROWS = 3;
const int COLS = 5;
....
for (row=1; row<=ROWS; ++row) {
     for (col=1; col<=COLS; ++col) {
          printf("* ");
     }
     printf("\n");
}
```

Another 2-D Figure

Print

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

```
const int ROWS = 5;
....
int row, col;
for (row=1; row<=ROWS; ++row) {
     for (col=1; col<=row; ++col) {
          printf("* ");
     }
     printf("\n");
}
```

Yet Another One

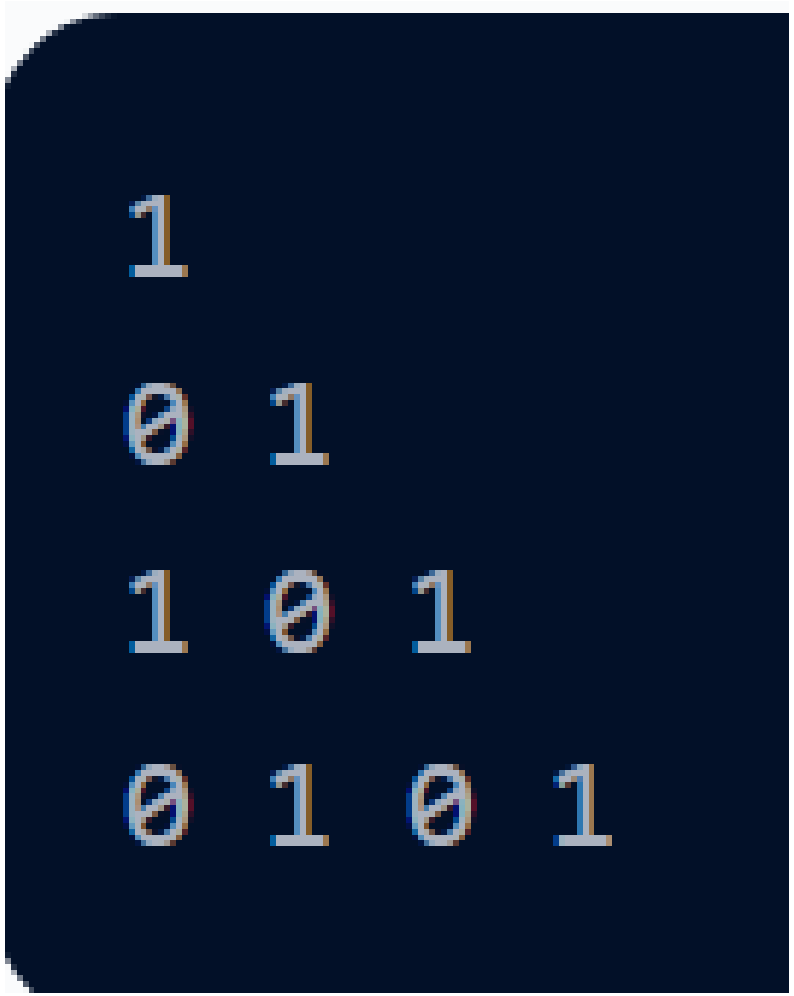
Print

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

```
const int ROWS = 5;

....
int row, col;
for (row=0; row<ROWS; ++row)   {
      for (col=1; col<=row; ++col)
            printf("   ");
      for (col=1; col<=ROWS-row; ++col)
            printf("* ");
      printf ("\n");
}
```

```
1.      int i,j,n;
2.      for (int i = 0; i < n; i++) {
3.          for (int j = 0; j <= i; j++) {
4.          //logic
5.          }
6.      }
```

- if we are at row number i and column number j, then based on the parity of (i+j), we can say:
- if (i+j) is even, then print 1
- if (i+j) is odd, then print 0



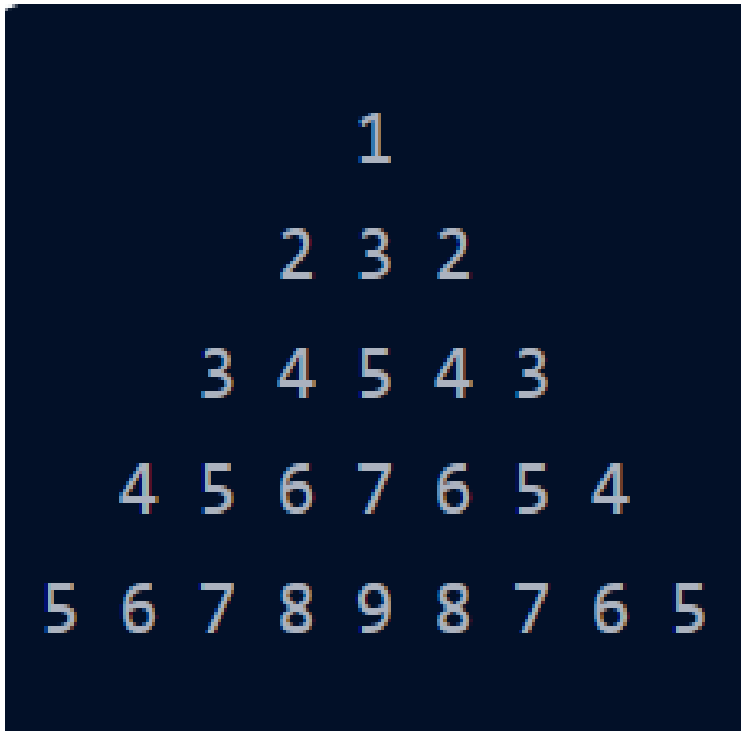
Full Pyramid of Numbers eg1.c

P1

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

P2

```
- - - -
2 - - -
4 3 - -
6 5 4 -
8 7 6 5
```



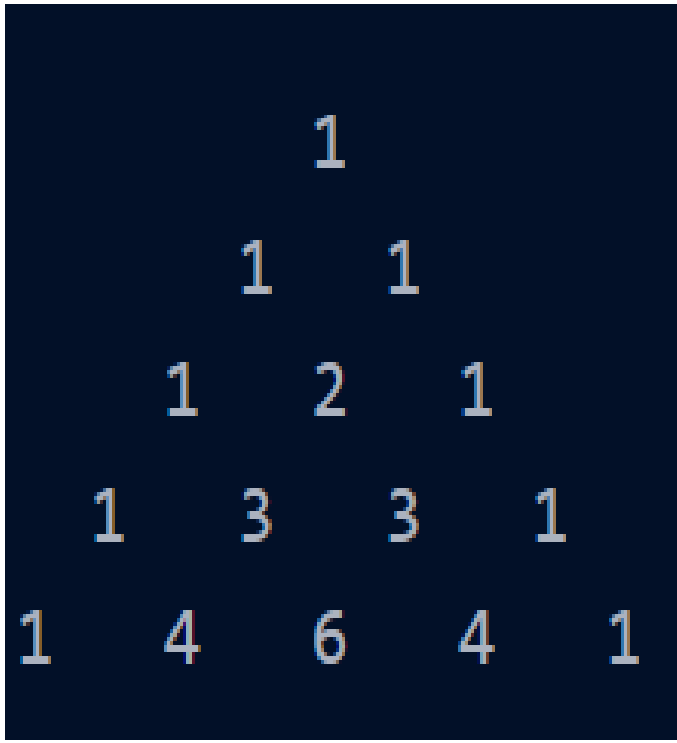
Pascal's Triangle eg2.c

- Pascal's Triangle is a triangular array of the binomial coefficients i.e. if we are row i and column, then the pattern will display:

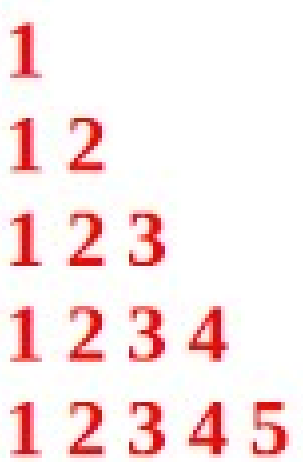
```
C(i,j) = i! / ( (i-j)! * j! )
```

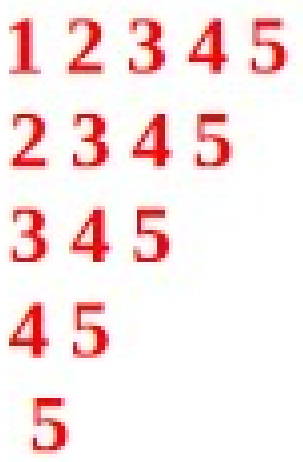
Now let us try to calculate C(i,j) from C(i, j-1):

```
     C(i, j)   = i! / ( (i-j)! * j! )
    C(i, j-1) = i! / ( (i - j + 1)! * (j-1)! )
    //We can derive the following expression
from the above two expressions.
    C(i, j) = C(i, j-1) * (i - j + 1) / j
```



Nested loops – loop *within* loop





Exercise 2:

- Write a C program that will read a decimal integer and will convert to equivalent to binary number

What is the output?

```
main(){
int i,j,n=6;
for(i=1;i<=n;i++){
  for(j=1;j<=i;j++){
    printf("*");
  }
  printf("\n");
}}
```

break and continue

- Break statement helps to break from the immediate local loop for certain condition.

```
main(){
int i,j;
for(i=1;i<=10;i++){
  if(i%6==0) break;    //if the number is div by 6
}
printf("i=%d",i);   //output i=6
}
```

Continue – skip to next iteration

```
main(){
int i,j;
for(i=1;i<=10;i++){
  if(i%3!=0) printf("i=%d\n",i);
  else continue;  //skip to next iteration
}}
//output – 1 2 4 5 7 8 10
```

Practice Problems (do each with both for and while loops separately)

1. Read in an integer N. Then print the sum of the squares of the first N natural numbers.
2. Read in an integer N. Then read in N numbers and print their maximum and second maximum (do not use arrays even if you know it)
3. Read in an integer N. Then read in N numbers and print the number of integers between 0 and 10 (including both), between 11 and 20, and > 20. (do not use arrays even if you know it)
4. Repeat 3, but this time print the average of the numbers in each range.
5. Read in a positive integer N. If the user enters a negative integer or 0, print a message asking the user to enter the integer again. When the user enters a positive integer N finally, find the sum of the logarithmic series ($\log_e(1+x)$) upto the first N terms
6. Read in an integer N. Then read in integers, and find the sum of the first N positive integers read. Ignore any negative integers or 0 read (so you may actually read in more than N integers, just find the sum with only the positive integers and stop when N such positive integers are read)
7. Read in characters until the '\n' character is typed. Count and print the number of lowercase letters, the number of uppercase letters, and the number of digits entered.

Arrays

- An array is a collection of one or more values of the same data type stored in contiguous memory locations.
- Arrays in C are classified into three types:
    - One-dimensional arrays
    - Two-dimensional arrays
    - Multi-dimensional arrays

Rules for Declaring One Dimensional Array in C

- Before using and accessing, we must declare the array variable.
- In an array, indexing starts from 0 and ends at size-1. For example, if we have arr[10] of size 10, then the indexing of elements ranges from 0 to 9.
- We must include data type and variable name while declaring one-dimensional arrays in C.
- We can initialize them explicitly when the declaration specifies array size within square brackets is not necessary.
- Each element of the array is stored at a contiguous memory location with a unique index number for accessing.