

C programming

MODULE 2

CSE Dept.

ASIET



APJ Abdul Kalam Technological University
Adi Shankara Institute of
Engineering and Technology, Kalady
Department of Computer Science & Engineering &
Information Technology





Module 2

Program Basics



syllabus

Basic structure of C program: Character set, Tokens, Identifiers in C, Variables and Data Types ,Constants, Console IO Operations, printf and scanf

Operators and Expressions: Expressions and Arithmetic Operators, Relational and Logical Operators, Conditional operator, size of operator, Assignment operators and Bitwise Operators. Operators Precedence

Control Flow Statements: If Statement, Switch Statement, Unconditional Branching using goto statement, While Loop, Do While Loop, For Loop, Break and Continue statements.(Simple programs covering control flow)

1. a. Compare between compiler and assembler. (3)
b. Mention any four keywords and their meaning. (2)

9. a. Draw the flowchart and develop the algorithm for finding the area of a triangle by reading three sides (5)
b. Explain the different datatypes in C. (5)

Answer any five questions, each carries 5 marks.

- 9 a) Explain any five kinds of operators in C. (5)
b) Draw a flowchart to find the sum of digits of an integer. (5)

1. a) Discuss various data types used in C with examples. (3 marks)
b) What do you understand by the term Keyword? (2 marks)

9. Differentiate Machine Language, Assembly Language and High level Language.

(10 marks)

Answer all questions, each carries 5 marks.

- 1 Differentiate between machine language, assembly language and high level languages? What is the difference between compiler and assembler? (5)

10 a) If more than one kind of operator is present in an expression, explain the order of precedence. (5)

1 What is a compiler? How does it differ from an interpreter? (5)

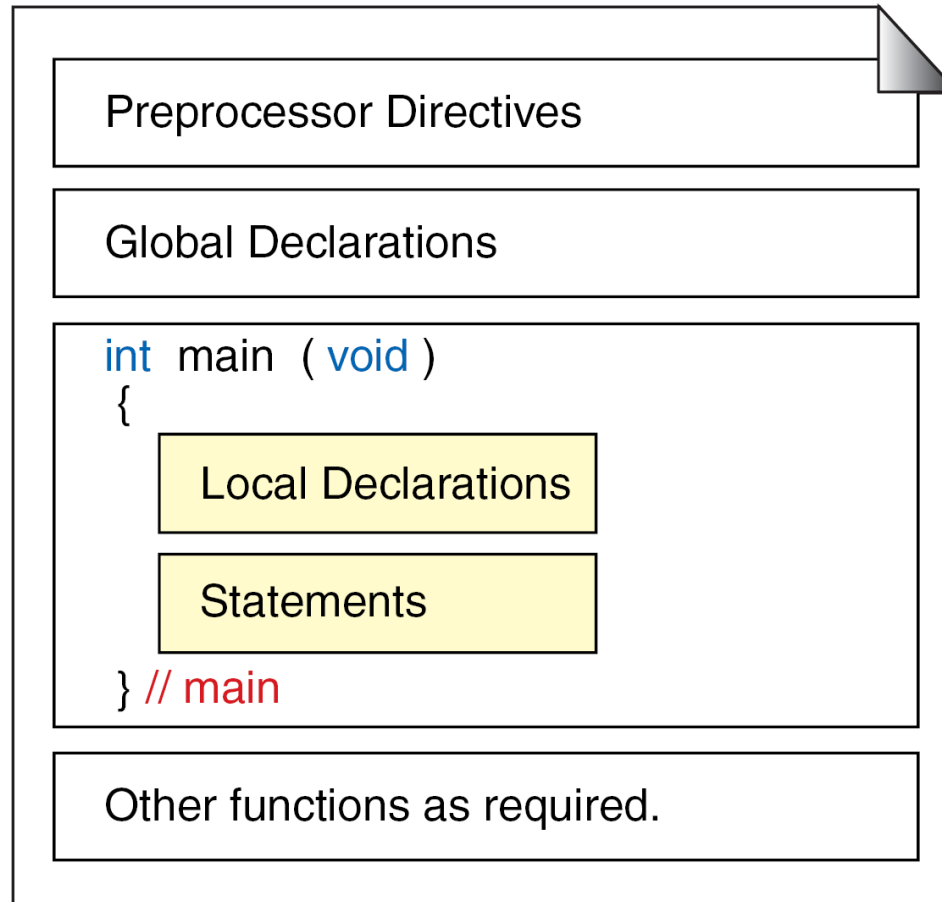
9 a) Differentiate between ++ i and i ++ with the help of examples. (4)

b) Explain different data types in C with examples. (6)

1 What are the advantages of High Level languages? How a high-level language is converted to Machine language? (5)

Structure of a C program

Documentations (Documentation Section)



```
/* Author: www.w3schools.in  
Date: 2018-04-28  
Description:  
Writes the words "Hello, World!" on the screen */  
#include<stdio.h>  
  
int main()  
{  
    printf("Hello, World!\n");  
    return 0;  
}
```

Documentation Section

`/* Comments */` multiline
comment

`//` single line comment

This is a comment block,
which is ignored by the
compiler.

Comment can be used
anywhere in the program
which will be helpful for
developers to understand
the existing code in the
future easily.

Preprocessor directives

Preprocessor directives

- The Preprocessor, as the name implies, is a program that processes the source code before it passes through the compiler.
- Preprocessor directives begin with the symbol `#` in column one and do not require a semicolon at the end.

Types of preprocessor directives

Preprocessor	Syntax/Description
Macro	Syntax: #define This macro defines constant value and can be any of the basic data types.
Header file inclusion	Syntax: #include <file_name> The source code of the file “file_name” is included in the main program at the specified place.

Macro Substitution directive

The general form is

#define identifier string

Examples

#define COUNT 100 (simple macro)

Area of a circle

```
#include <stdio.h>
#define PI 3.14
void main()
{
    int radius;
    float area;
    printf("enter radius");
    scanf("%d",&radius);
    area=PI*radius*radius;
    printf("area is %f ",area);

}
```

File Inclusion directive

This is achieved by

`#include "filename"` or `#include <filename>`

Examples

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

HEADER FILES IN C

- Pre-processor statements begin with a # character.
- In this case, the pre-processor statement we have is a #include statement.
- Included files are known as *header files* and by convention have the extension .h.

- `<math.h>`

Defines common mathematical functions.

- `<stdio.h>`

Defines core input and output functions

- `<stdlib.h>`

Defines numeric conversion functions, pseudo-random numbers generation functions, memory allocation, process control functions

- `<string.h>`

Defines string-handling functions



Building block of C

1. Character set
2. TOfkens
 1. Identifiers
 2. Keywords
 3. String
 4. Constants & variables
 5. String
 6. Special symbol
 7. Operator
3. data types
4. Expressions
5. Statements, Input and Output statements

1 .Character set

- A character denotes any alphabet, digit or special symbol used to represent information.

Alphabets	A, B,, Y, Z a, b,, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ' ! @ # % ^ & * () _ - + = \ { } [] : ; " ' < > , . ? /

1. Character set

- White space characters

Tab, newline, blank space

- Each character in `c` identified with its ASCII value

2. Identifiers

- *identifiers are names that are given to various program elements, such as variables, functions and arrays.*
- Identifiers consist of letters and digits, in any order, except that ***the first character must be a letter or underscore.***
- **Both** upper- and lowercase letters are permitted
- Upper- and lowercase letters are not interchangeable (i.e., an uppercase letter is **not** equivalent to the corresponding lowercase letter.)
- An identifier can be arbitrarily long.
- Some implementations of C recognize only the first eight characters, though most implementations recognize more (typically, **31 characters**).

The following names are valid identifiers.

x

y12

sum_1

_temperature

names

area

tax_rate

TABLE

The following names are *not* valid identifiers for the reasons stated.

4th

The first character must be a letter.

"x"

Illegal characters (").

order-no

Illegal character (-).

error flag

Illegal character (blank space).

3. keywords

- There are certain reserved words, called ***keywords, that have standard, predefined meanings in C.***
- They cannot be used as programmer-defined identifiers.
- keywords are all lowercase

C Keywords

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

4. Variables

- Variable names are names given to locations in memory.
- These locations can contain **integer, real or character constants**.
- A particular type of variable can hold only the same type of constant.
- For example,
 - an integer variable can hold only an integer constant
 - a real variable can hold only a real constant
 - and a character variable can hold only a character constant.

Rules for Constructing Variable Names

1. A variable name is any combination of alphabets, digits or underscores.
2. Some compilers allow variable names whose length could be up to 247 characters. Still, it would be safer to stick to the rule of 31 characters.
3. The first character in the variable name must be an alphabet or underscore.
4. No commas or blanks are allowed within a variable name.
5. No special symbol other than an underscore (as in **gross_sal**) can be used in a variable name.
6. Uppercase and lowercase are taken differently

Which of the following are valid variables

Determine which of the following are valid identifiers. If invalid, explain why.

(a) record1

(e) \$tax

(h) name_and_address

(b) 1record

(f) name

(i) name-and-address

(c) file_3

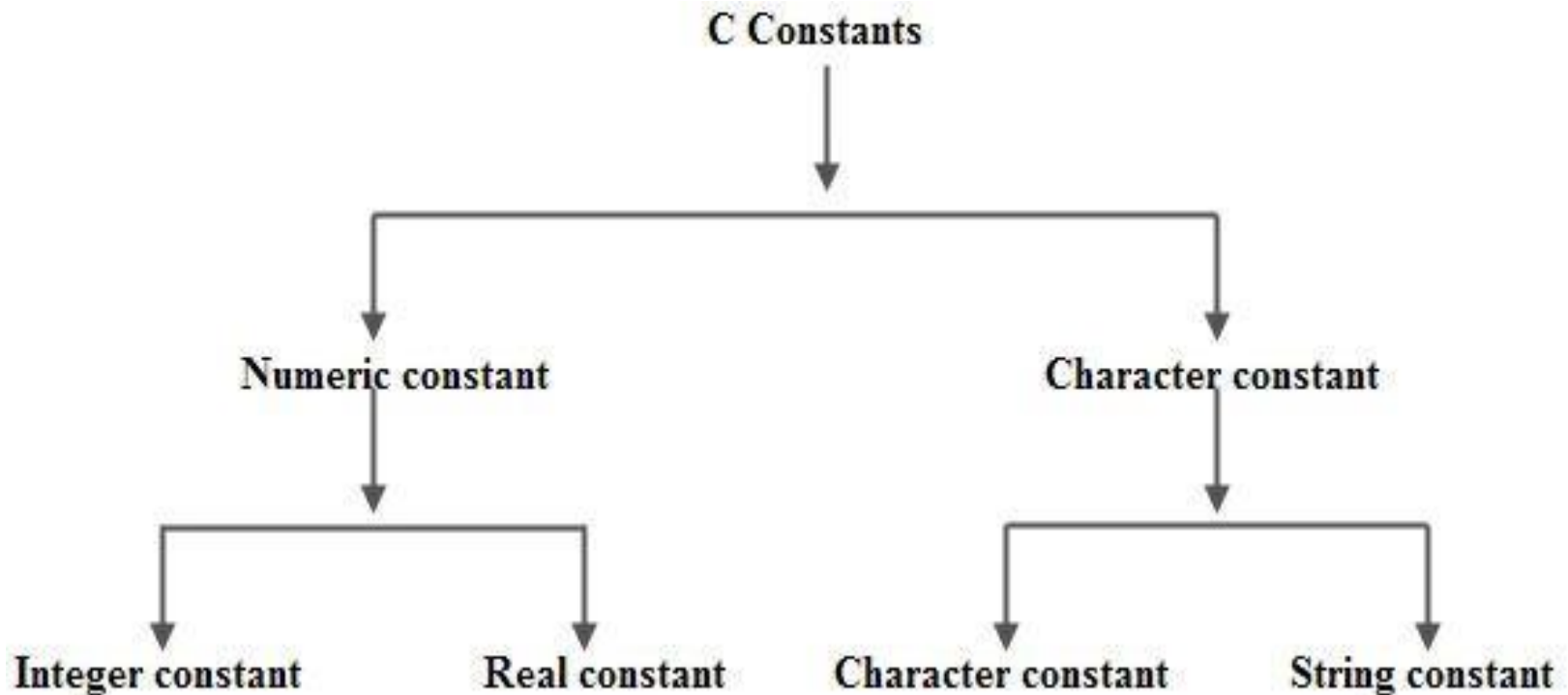
(g) name and address

(j) 123.45.6789

(d) return

5.constants

- Fixed value that do not change during the execution of program.



1 Integer Constants

1. decimal,
2. octal (Value starts with 0)
3. hexadecimal number (Value starts with 0x).

2 Real constants

Real constants

- The default form of real constant is double and it must have a decimal point.
- You can represent the negative numbers in real constants.
- It may be in fractional form or exponential form.
- Ex: 3.45, -2.58, 0.3E-5 (equal to 0.3×10^{-5})

Representation	Value	Type
0	0.0	double
6.77	6.77	double
-6.0f	-6.0	float
3.1415926536L	3.1415926536	long double

- **Character Constants**
- Character Constants must be enclosed with in single quotes.
- **A string constant**
- is a sequence of characters enclosed in a double quotes.

Examples:

"programming9" // a full string with 12 characters


```
const data_type variable_name;
```

```
const float pi =3.14;
```

(or)

```
data_type const variable_name;
```

```
float const pi =3.14;
```

You can also define as follows

```
#define pi 3.14
```

```
#include<stdio.h>
void main()
{
const int a=10; //Decimal
int const b= 012; //Octal
const int c=0xA; //Hexadecimal
printf(" %d \n %d \n %d",a,b,c);
}
```

OUTPUT

10

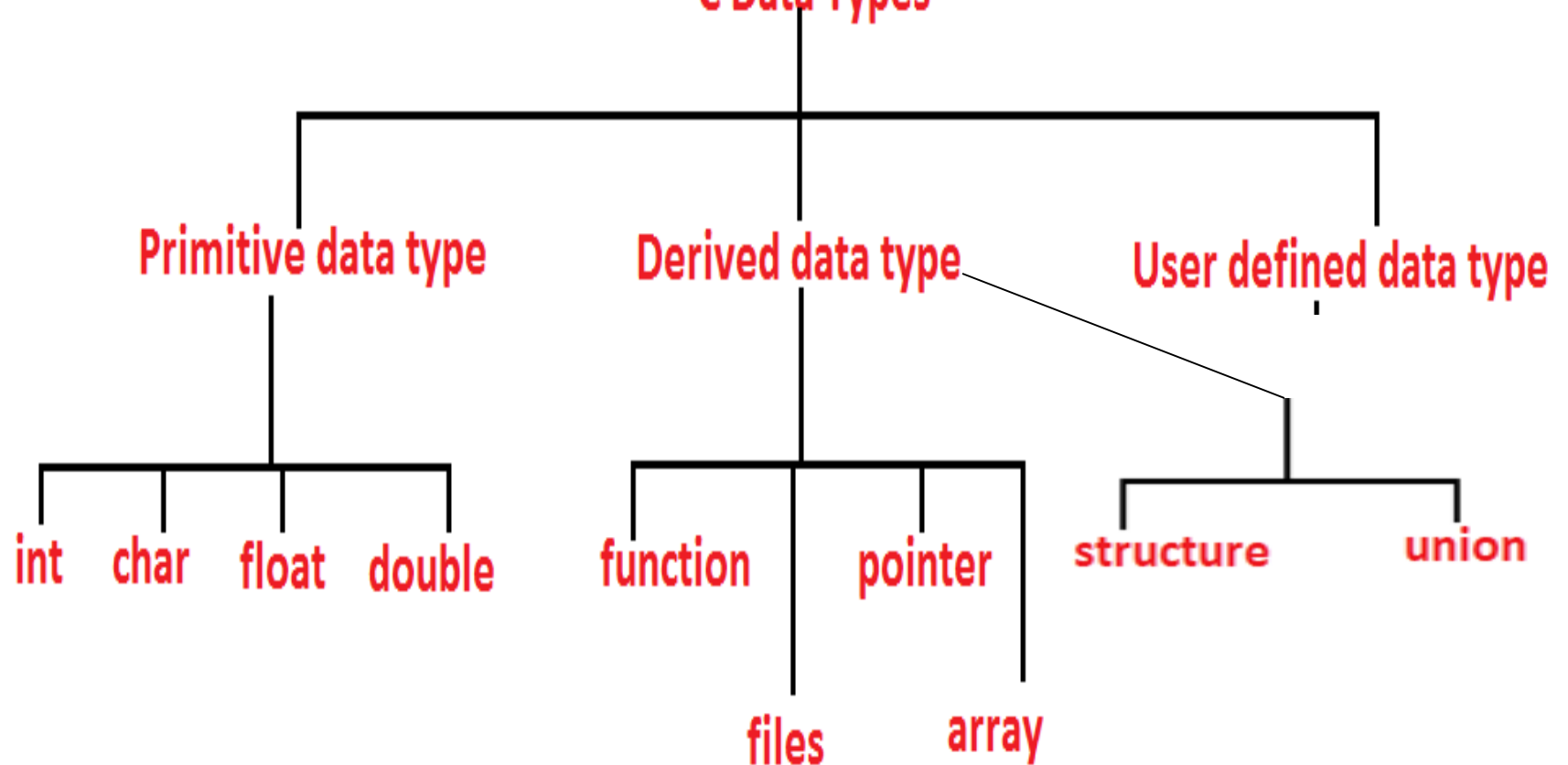
10

10

DATA TYPES

- Data type determines the type of data a particular variable can hold
- Memory requirements for each data type may vary from one C compiler to another.

C Data Types



- **Primitive data types** are the first form – the basic data types (int, char, float, double).
- **Derived data types** are a derivative of primitive data types known as arrays, pointer
- **User defined** data types are those data types which are defined by the user/programmer himself.

type	Format Specifier	Memory requirement
char	%c	1 byte
int	%d	2 bytes traditional compiler 4 bytes in gcc
float	%f	4 bytes
double	%lf	8 bytes

The four basic data types are

- char
- int
- float
- double

INTEGER

- These are whole numbers, both positive and negative.
- The keyword used to define integers is,

`int`

- An example of an integer value is 32.
- An example of declaring an integer variable called sum is,
 - `int sum;`
 - `sum = 20;`

FLOATING POINT

- These are numbers which contain fractional parts, both positive and negative.
- The keyword used to define float variables is,
float
- An example of a float value is 34.12345.stored as scientific notation 0.3412345×10^2
- In E notation 0.3412345E2
- Precision is 7digit

DOUBLE

- The keyword used to define double variables is, *double*

An example of a double value is .30E2.

An example of declaring a double variable called **big** is,

– *double big;*

– *big = 312E+7;*

- Precision is 15 digit

CHARACTER

- These are single characters.
- The keyword used to define character variables is, *char*
- **An example of declaring a character variable called letter is,**
 - *Char letter;*
 - *letter = 'A';*
- Note the assignment of the character *A* to the variable *letter* is done by enclosing the value in single quotes.
- **Remember the golden rule: Single character - Use single quotes.**

Data type qualifiers

- Qualifiers alters the meaning of base data types to yield a new data type.
- They are used to improve the program execution speed
 1. **Size qualifiers**
 2. **Sign qualifiers**

Size qualifiers

- Size qualifiers alters the size of a basic type. There are two size qualifiers
 - *long*
 - *short*.
 - For integer data types, there are three sizes:
 - int, and two additional sizes called **long** and **short**, which are declared as long int and short int.
- The long is intended to provide a larger size of integer, and short is intended to provide a smaller size of integer.

Sign qualifiers

- 2 types
 1. Signed
 - can store both positive and negative values
 2. unsigned
 - can store only positive values
- *signed* is default one,
i.e., declaring `int a;` is same as
`signed int a;`

unsigned

- if a variable needs to hold positive value only, *unsigned* data types are used.
- Eg
- `unsigned int positiveInteger;`

Data type	Bytes occupied in RAM	Value range	Range of values referred
unsigned char	1 byte	0 to $2^8 - 1$	0 to 255
short int	2 bytes	-2^7 to $2^7 - 1$	-128 to 127
unsigned short int	2 bytes	0 to $2^8 - 1$	0 to 255
unsigned int	4 bytes	0 to $2^{16} - 1$	0 to 65535
long int	4 bytes	-2^{31} to $2^{31} - 1$	-2,147,483,648 to 2,147,483,647
unsigned long int	4 bytes	0 to $2^{32} - 1$	0 to 4,294,967,295
long double	10 bytes	-2^{79} to $2^{79} - 1$	3.4e-4932 to 1.1e+4932



Input output statements in c (Formatted I/O)

Input statement

Input data can be entered into the computer from a standard input device by means of the C library function ***scanf***.

scanf(control string, arg1, arg2, . . . , argn)

control string refers to a string containing certain required formatting information,

arg1, arg2, . . . argn are arguments that represent the individual input data items.

scanf Syntax:

The **control string** characters must begin with a **percent sign (%)**.

Followed by **conversion character** that indicate the **type of the item**

Argument refers to the address of data item in memory

Always preceded with **&**

- `Scanf("%d", &a)` reading a integer value to a variable **a**
- `Scanf("%c", &s)` reading character value to a variable **s**

Commonly Used Conversion Characters for Data Input

<i>Conversion Character</i>	<i>Meaning</i>
c	data item is a single character
d	data item is a decimal integer
e	data item is a floating-point value
f	data item is a floating-point value
g	data item is a floating-point value
h	data item is a short integer
i	data item is a decimal, hexadecimal or octal integer
o	data item is an octal integer
s	data item is a string followed by a whitespace character (the null character \0 will automatically be added at the end)
u	data item is an unsigned decimal integer
x	data item is a hexadecimal integer
[. . .]	data item is a string which may include whitespace characters (. . .)

Output statement

- Output data can be written from the computer onto a standard output device using the library function **printf**.

`printf(control string, arg1, arg2, . . . , argn)`

or

`printf("message")`

Here argument **not preceded with &**

Eg `printf("%d", a)`

printing a integer value of variable **a**

Operators and Expressions

Operators

- The symbols which are used to perform logical and mathematical operations in a C program are called C operators.

Expression

- Operators, constants and variables are combined together to form expressions.
- Consider the expression $A + B * 5$.
- where, $+$, $*$ are operators, A, B are variables, 5 is constant and $A + B * 5$ is an expression.

- Unary operator
 - operators that act upon a single operand
 - Unary +
 - Unary -
 - Increment ++
 - Decrement --
 - Size of `sizeof()`
- Binary operator
- Ternary operator
 - Condition? true_value: false_value;
 - `(A > 100 ? 0:1)`;

Types of Operators in C

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators

Arithmetic operators

Operator	Meaning	Example
+	Addition Operator	$10 + 20 = 30$
-	Subtraction Operator	$20 - 10 = 10$
*	Multiplication Operator	$20 * 10 = 200$
/	Division Operator	$20 / 10 = 2$
%	Modulo Operator	$20 \% 6 = 2$

Types of Operators	Description
Arithmetic_operators	These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus
Assignment_operators	These are used to assign the values for the variables in C programs.
Relational operators	These operators are used to compare the value of two variables.
Logical operators	These operators are used to perform logical operations on the given two variables.
Bit wise operators	These operators are used to perform bit operations on given two variables.
Conditional (ternary) operators	Conditional operators return one value if condition is true and returns another value if condition is false.
Increment/decrement operators	These operators are used to either increase or decrease the value of the variable by one.
Special operators	&, *, sizeof() and ternary operators.

Types of Operators in C

1. Arithmetic operators
2. **Assignment operators**
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

Assignment Operators

1. Simple assignment operator (Example: =)
2. Compound assignment operators (Example: +=, -=, *=, /=, %=, &=, ^=)

Operators	Example/Description
=	sum = 10; 10 is assigned to variable sum
+=	sum += 10; This is same as sum = sum + 10
-=	sum -= 10; This is same as sum = sum - 10
*=	sum *= 10; This is same as sum = sum * 10
/=	sum /= 10; This is same as sum = sum / 10
%=	sum %= 10; This is same as sum = sum % 10
&=	sum&=10; This is same as sum = sum & 10
^=	sum ^= 10; This is same as sum = sum ^ 10

Types of Operators in C

1. Arithmetic operators
2. Assignment operators
3. **Relational operators**
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

Relational operators

Operators	Example/Description
>	$x > y$ (x is greater than y)
<	$x < y$ (x is less than y)
>=	$x \geq y$ (x is greater than or equal to y)
<=	$x \leq y$ (x is less than or equal to y)
==	$x == y$ (x is equal to y)
!=	$x != y$ (x is not equal to y)

Types of Operators in C

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. **Logical operators**
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

Logical operators

Operators	Example/Description
&& (logical AND)	<code>(x>5)&&(y<5)</code> It returns true when both conditions are true
(logical OR)	<code>(x>=10) (y>=10)</code> It returns true when at-least one of the condition is true
! (logical NOT)	<code>!((x>5)&&(y<5))</code> It reverses the state of the operand “ <code>((x>5) && (y<5))</code> ” If “ <code>((x>5) && (y<5))</code> ” is true, logical NOT operator makes it false

Types of Operators in C

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. **Bit wise operators**
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

Bit-wise operators

1. $\&$ – Bitwise AND
2. $|$ – Bitwise OR
3. \sim – Bitwise NOT
4. \wedge – XOR
5. \ll – Left Shift
6. \gg – Right Shift

Bit-wise operators

Example

Consider $x=40$ and $y=80$. Binary form of these values are given below.

X = 00101000

y= 01010000

All bit wise operations for x and y are given below.

- [illegible]

```
#include <stdio.h>
int main()
{
int a = 12, b = 25;
printf("Output = %d", a&b);
return 0;
}
```

Output

Output = 8

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

00001100 &

00011001

00001000 = 8 (In decimal)

Types of Operators in C

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. **Conditional operators (ternary operators)**
7. Increment/decrement operators
8. Special operators

Conditional operators

(ternary operator)

- Conditional operators return one value if condition is true and returns another value if condition is false.
- This operator is also called as ternary operator.

Syntax : (Condition? true_value: false_value);

Example : A > 100 ? 0 : 1;

Program to find largest of 2 numbers using ternary operator

```
#include <stdio.h>
int main()
{
    int num1, num2, max;
    printf("Enter two numbers: ");
    scanf("%d%d", &num1, &num2);
    max = (num1 > num2) ? num1 : num2;
    printf("Maximum is %d", max);
    return 0;
}
```

Types of Operators in C

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. **Increment/decrement operators**
8. Special operators

Increment/decrement Operators

- Increment operators are used to increase the value of the variable by one
- decrement operators are used to decrease the value of the variable by one in C programs.
- Example:

Increment operator : `++ i ;` `i ++ ;`

Decrement operator : `-- i ;` `i -- ;`

A short note about ++

- ++i means increment i then use it
- i++ means use i then increment it

```
int i= 6;  
printf ("%d\n",i++); /* Prints 6 sets i to 7 */
```

Note this important difference

```
int i= 6;  
printf ("%d\n",++i); /* prints 7 and sets i to 7 */
```

It is easy to confuse yourself and others with the difference between ++i and i++ - it is best to use them only in simple ways.

All of the above also applies to --.



```
#include <stdio.h>
int main()
{
    int i= 6;
    i++;
    printf ("%d\n",i);
    return 0;
}
```



```
#include <stdio.h>
int main()
{
    int i= 6;
    i++;
    printf ("%d\n",i);
    return 0;
}
```

Output
7



```
#include <stdio.h>

int main()
{int i= 6,a;
a=i++;
printf ("%d\n",a);
return 0;
}
```



```
#include <stdio.h>

int main()
{int i= 6,a;
a=i++;
printf ("%d\n",a);
return 0;
}
```

Output
6

Types of Operators in C

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. **Special operators**

Example

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

```
int main()
```

```
{
```

```
    int q;
```

```
    q = 50;
```

```
    printf("%d", sizeof(q));/* display the q's sizein bytes*/
```

```
    return 0;
```

```
}
```

C Data types / storage Size	Range
char / 1	-127 to 127
int / 2	-32,767 to 32,767
float / 4	1E-37 to 1E+37 with six digits of precision
double / 8	1E-37 to 1E+37 with ten digits of precision
long double / 10	1E-37 to 1E+37 with ten digits of precision
long int / 4	-2,147,483,647 to 2,147,483,647
short int / 2	-32,767 to 32,767
unsigned short int / 2	0 to 65,535
signed short int / 2	-32,767 to 32,767
long long int / 8	$-(2^{\text{power}(63)} - 1)$ to $2^{\text{power}(63)} - 1$
signed long int / 4	-2,147,483,647 to 2,147,483,647
unsigned long int / 4	0 to 4,294,967,295
unsigned long long int / 8	$2^{\text{power}(64)} - 1$

Module I

Introduction to C Language:

- i. Preprocessor directives
- ii. Header files
- iii. Data types and qualifiers
- iv. Operators and expressions
- v. **Data input and output**
- vi. Control statements

Operator precedence

OPERATOR	TYPE	ASSOCIATIVITY
() [] . ->		left-to-right
++ -- + - ! ~ (type) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= = <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

Data Input and Output

Input and output functions

- In C Input and output functions are of 2 types
 1. Formatted
 2. unformatted

Formatted functions

- Such functions read and write all type of data values
- They require conversion symbol to identify the data type
- Formatted functions return the value after execution which is equal to the number of variables successfully read/written

Eg:

`printf,scanf`

Format Specifier

- `%d` - the value of an integer variable.
- `%c` - character,
- `%f` - float variable,
- `%s` - string variable,
- `%lf` - double
- `%x` - hexadecimal variable
- `“\n”` - generate a newline

```
#include <stdio.h>

int main()
{
    char ch;
    char str[100];
    printf("Enter any character \n");
    scanf("%c", &ch);
    printf("Entered character is %c \n",
        ch);
    printf("Enter any string ( upto 100
        character ) \n");
    scanf("%s", &str);
    printf("Entered string is %s \n", str);
}
```

Output :

Enter any character

a

Entered character is a

Enter any string (upto
100 character)

hai

Entered string is hai

Unformatted I/O

- **3 types**
 - 1. Character I/O**
 - 2. String I/O**
 - 3. File I/O**

Unformatted I/O

- **3 types**
 - 1. Character I/O**
 - `getch()`, `putch()`
 - `getchar()`, `putchar()`
 - `getche()`
 - 2. String I/O**
 - `gets()`, `puts()`
 - 3. File I/O**
 - `getc()`, `putc()`

Un Formatted functions

- Such functions works with one type of data values
- They do not require conversion symbol to identify the data type because they work only with character data type.

Character I/O

- `getchar()`
- The `getchar()` function reads a character from keyboard
- This function reads only single character at a time
- It has following form

`character variable =getchar();`

where character variable is some previously defined character variable

Character I/O

- `putchar()`
- The `putchar()` function prints the character passed to it on the screen and returns the same character.
- This function puts only single character at a time
- It has following format

`putchar(character variable);`

where character variable is some previously defined character variable

Pgm Read and print a character using Getchar() putchar()

```
#include <stdio.h>
void main( )
{
    char c;
    printf("Enter a character");
    c=getchar();
    putchar(c);

}
```


String I/O

- **gets()**
 - to read a string
- **puts()**
 - To print a string

File I/O



- Similar to `getchar()` and `putchar()` functions
- `getc()`
 - to read a character from a file
- `putc()`
 - to write a character to a file

ASCII character set

- Most computers, and virtually all personal computers, make use of the **ASCII** (i.e., **American Standard Code for Information Interchange**) character set, in which each individual character is numerically encoded with its own unique 7-bit combination (hence a total of $2^7 = \mathbf{128}$ **different characters**).
- contains the ASCII character set, showing the decimal equivalent of the **7 bits that represent each character**.

Table 2-1 The ASCII Character Set

<i>ASCII Value</i>	<i>Character</i>	<i>ASCII Value</i>	<i>Character</i>	<i>ASCII Value</i>	<i>Character</i>	<i>ASCII Value</i>	<i>Character</i>
0	NUL	32	(blank)	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

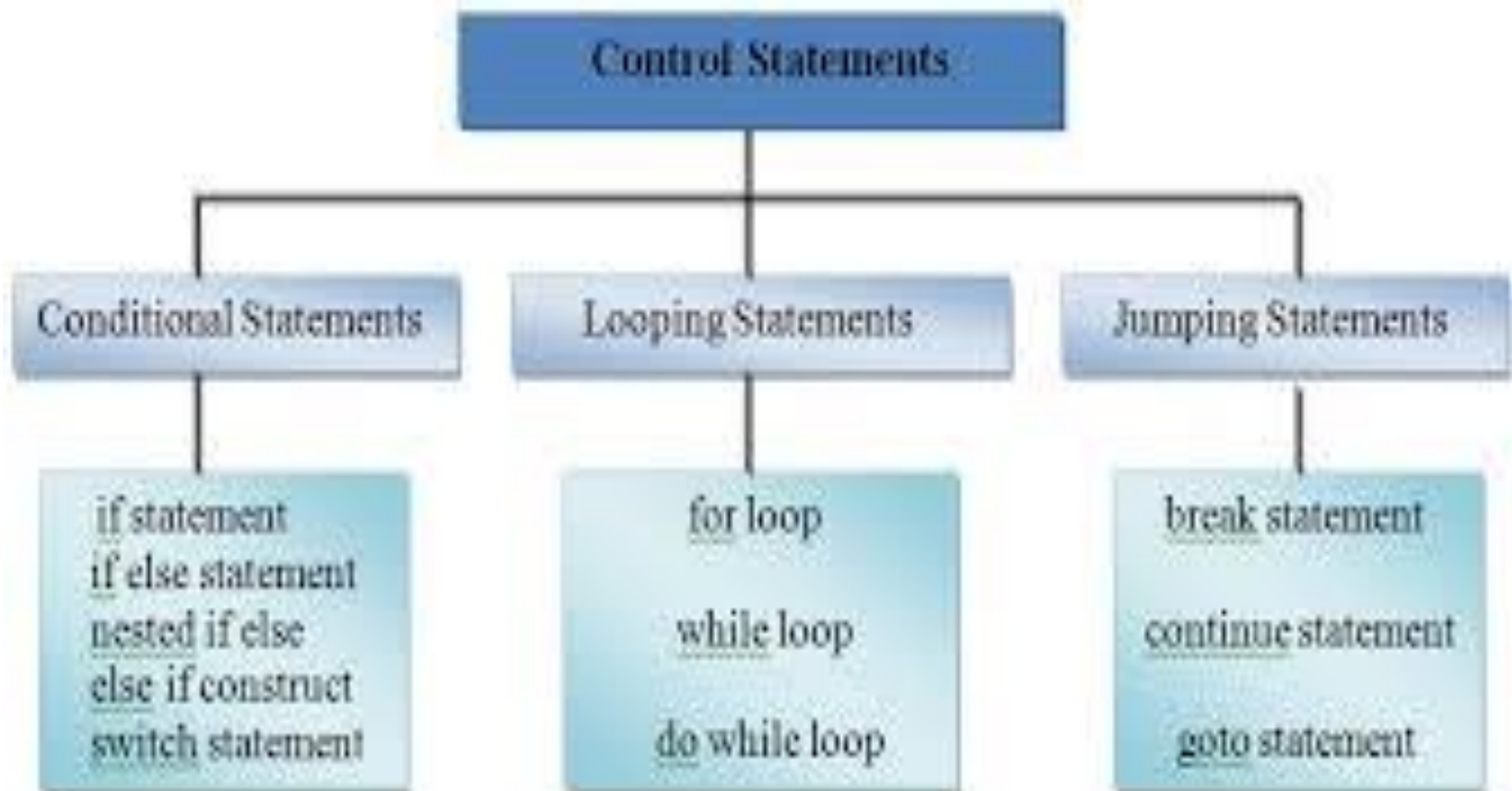
The first 32 characters and the last character are **control characters**. Usually, they are not displayed. However, some versions of C (some computers) support special graphics characters for these ASCII values. For example, 001 may represent the character , 002 may represent , and so on.

Module I

Introduction to C Language:

- i. Preprocessor directives
- ii. Header files
- iii. Data types and qualifiers
- iv. Operators and expressions
- v. Data input and output
- vi. Control statements**

Control statements



Decision Control Statements

- If condition is true group of statements are executed.
- If condition is false, then else part statements are executed.

3 types

1. if statements
2. if else statements
3. nested if statements

<p>if</p>	<p>Syntax: if (condition) { Statements; }</p> <p>Description: In these type of statements, if condition is true, then respective block of code is executed.</p>
<p>if...else</p>	<p>Syntax: if (condition) { Statement1; Statement2; } else { Statement3; Statement4; }</p> <p>Description: In these type of statements, group of statements are executed when condition is true. If condition is false, then else part statements are executed.</p>
<p>nested if</p>	<p>Syntax: if (condition1){ Statement1; } else if(condition2) { Statement2; } else Statement 3;</p> <p>Description: If condition 1 is false, then condition 2 is checked and statements are executed if it is true. If condition 2 also gets failure, then else part is executed.</p>

The if Statement

- The *if statement* has the following syntax:

if (*condition*)



statement1;

statement2;

The *condition* must be a boolean expression.

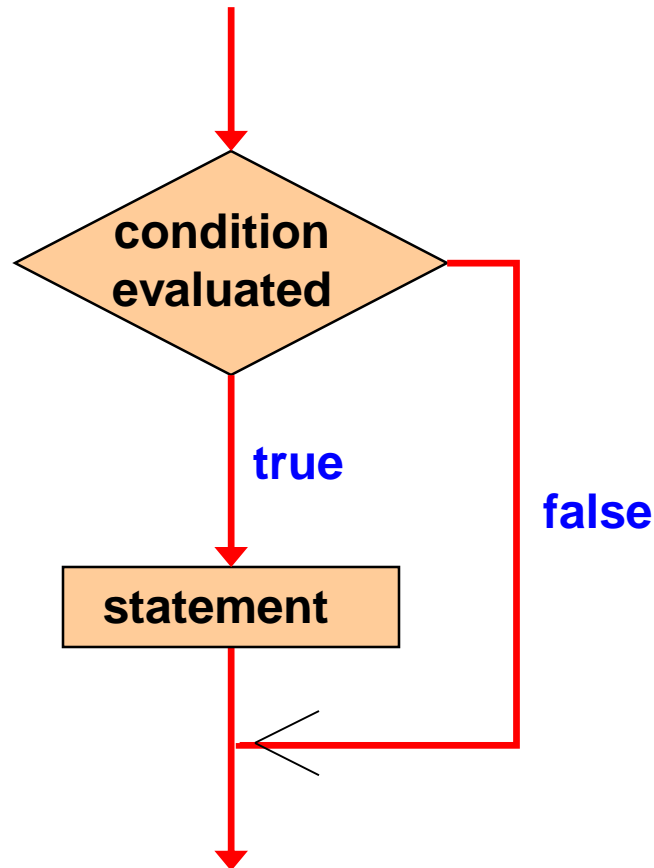
It must evaluate to either true or false.

If the *condition* is true, the *statement* is executed.

If it is false, the *statement* is skipped.



Logic of an if statement



Predict the output

```
#include<stdio.h>
main()
{
    int a=10;
    int b=20;
    if(a<b)
    {
        printf("asiet");
    }
    printf("cs department");
}
```

output

asiet

cs department

Predict the output

```
#include<stdio.h>
main()
{
    int a=10;
    int b=20;
    if(a<b)
    {    printf("welcome ");
        printf("asiet");
    }
    printf("cs department");
}
```

output

welcome

asiet

cs department

Predict the output

```
#include<stdio.h>
main()
{
    int a=10;
    int b=20;
    if(a>b)
    {    printf("welcome ");
        printf("asiet");
    }
    printf("cs department");
}
```

Out put
cs department

```
#include<stdio.h>
int main()
{
    if(8)
    {
        printf("asiet\n");
    }
    printf("out");
return 0;
}
```


output

asiet

out

- Non zero number always treated as truth
- Zero is always treated as falsity

```
#include<stdio.h>
int main()
{
    if(0)
    {
        printf("asiet\n");
    }
    printf("exit");
return 0;
}
```

Out put
exit

Pgm to Calculate the absolute value

```
#include<stdio.h>
void main()
{
    int a;
    printf("Enter number");
    scanf("%d",&a);
    if(a<0)
    {
        a= -a;
    }
    printf("Absolute value is%d",a);
}
```

if-else statement

If (condition)

```
{  
    statement 1;  
}
```

else

```
{  
    statement 2;  
}
```

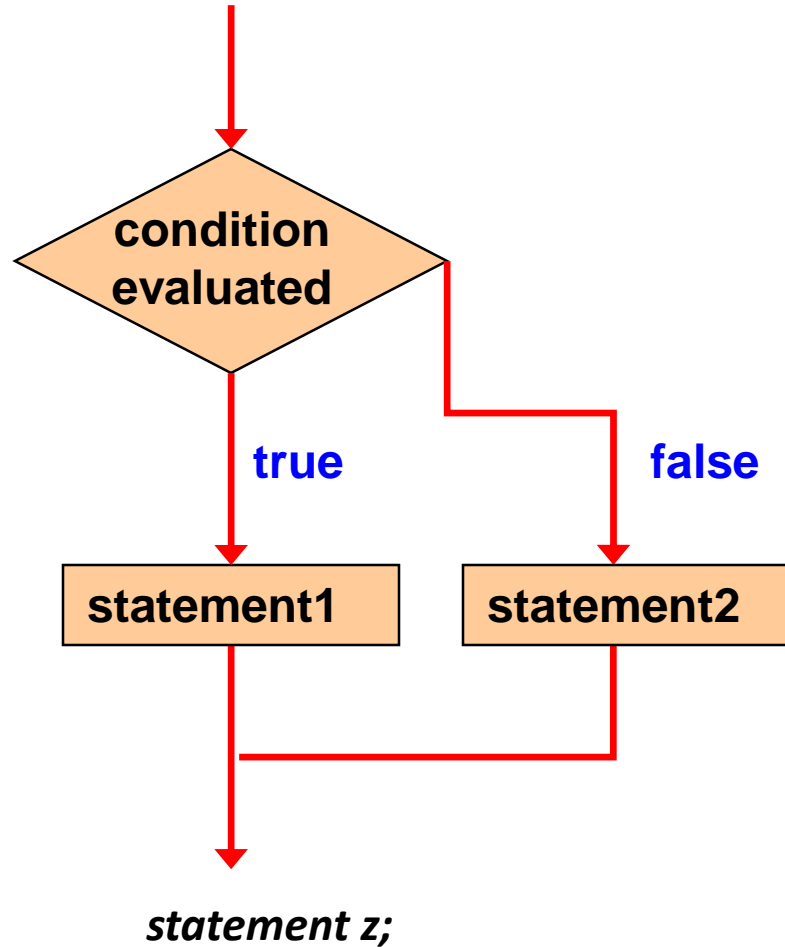
statement z;

If the *condition* is true,
statement1 is executed;

if the condition is false,
statement2 is executed

One or the other will be executed,
but not both

Logic of an if-else statement



Greatest of two numbers

```
#include<stdio.h>
void main()
{
    int a,b;
    printf("Enter numbers a and b");
    scanf("%d%d",&a,&b);

    if(a>b)
    {
        printf("The number %d is greater",a );
    }
    else
    {
        printf("The number %d is greater",b );
    }
}
```

Exercise

1. Program to check whether a given number is odd or even
2. Write a program to get marks for 3 subjects and declare the result. If the marks ≥ 35 in all the subjects the student passes else fails

Check whether the number is odd or even

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


NESTED IF

- It is possible to include if...else statement(s) inside the body of another if...else statement.

```
If(condition1)
{
    if(condition2)
    {
        statement1;
    }
    else
    {
        statement2;
    }
}
else
{
    Statement3;
}
```

**Nested if
SYNTAX:**

```
If(condition1)
```

```
{
```

```
    if(condition2)
```

```
    {
```

```
        statement1;
```

```
    }
```

```
    else
```

```
    {
```

```
        statement2;
```

```
    }
```

```
}
```

```
else
```

```
{
```

```
Statement3;
```

```
}
```

**Nested if
SYNTAX:**

```
If(condition1)
```

```
{
```

```
statement1;
```

```
}
```

```
else
```

```
{
```

```
    if(condition2)
```

```
    {
```

```
        statement2;
```

```
    }
```

```
    else
```

```
    {
```

```
        statement3;
```

```
    }
```

```
}
```

Nested if SYNTAX:

Pgm to check two numbers are equal ,if not display the greatest number

```
#include<stdio.h>
void main()
{
    int a,b;
    printf("Enter numbers a and b");
    scanf("%d%d",&a,&b);

    if(a==b)
    {
        printf("The numbers are equal");
    }
    else
    {
        if (a>b)
        {
            printf("The number %d is greater",a );
        }
        else
        {
            printf("The number %d is greater",b );
        }
    }
}
```

If else ladder- syntax

- Used When we need to select any one of the many alternatives.
- **ie when multiple decisions are involved .**

```
if(condition1)
```

```
{ // code to be executed if condition1 is true }
```

```
else if(condition2)
```

```
{ // code to be executed if condition2 is true }
```

```
else if(condition3)
```

```
{ // code to be executed if condition3 is true }
```

```
...
```

```
else if(condition n)
```

```
{
```

```
{ // code to be executed if conditionn is true }
```

```
}
```

```
else
```

```
{ // code to be executed if all the conditions are false }
```

Greatest of three numbers

```
#include<stdio.h>
void main()
{
    int a,b,c;
    printf("Enter numbers a,b and c");
    scanf("%d%d%d",&a,&b,&c);

    if((a>b) && (a>c))
    {
        printf("The number %d is greater",a );
    }
    else if((b>a) && (b>c))
    {
        printf("The number %d is greater",b );
    }
    else
    {
        printf("The number %d is greater",c );
    }
}
```

Exercise

- The variables **x** and **y** refer to numbers. Write a **code segment** that prompts the user for an arithmetic operator and prints the value obtained by applying that operator to **x** and **y**.

Program

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

```
void main()
```

```
{
```

```
    int x,y,sum,sub,mul;
```

```
    float div;
```

```
    char op;
```

```
    printf("enter operator");
```

```
    scanf("%c",&op);
```

```
    printf("enter x and y");
```

```
    scanf("%d%d",&x,&y);
```

```
    if(op=='+')
```

```
    {
```

```
        sum=x+y;
```

```
        printf("sum is %d",sum);
```

```
    }
```

```
    else if(op=='-')
```

```
    {
```

```
        sub=x-y;
```

```
        printf(" subtracted value is %d",sub);
```

```
    }
```

```
    else if(op=='*')
```

```
    {
```

```
        mul=x*y;
```

```
        printf(" multiplied value is %d",mul);
```

```
    }
```

```
    else if(op=='/')
```

```
    {
```

```
        div=x/y;
```

```
        printf(" divided value is %f",div);
```

```
    }
```

```
    else
```

```
        printf("Invalid operator");
```

```
    }
```

Exercise

- While purchasing certain items, a discount of 10% is offered if the quantity purchased is more than 1000.

If quantity and price per item are input through the keyboard, write a program to calculate the total expenses.

```
void main( )      /* Calculation of total expenses */
{
    int qty, dis = 0 ;
    float rate, tot ;
    printf ( "Enter quantity and rate " ) ;
    scanf ( "%d %f", &qty, &rate) ;
    if (qty>1000)
        dis=10;
    tot=(qty*rate)-(qty*rate*dis/100) ;
    printf("Total expenses=Rs. %f",tot);
}
```

```
Enter quantity and rate 1200 15.50
Total expenses = Rs. 16740.000000
Enter quantity and rate 200 15.50
Total expenses = Rs. 3100.000000
```

Exercise

- The current year and the year in which the employee joined the organization are entered through the keyboard.

If the number of years for which the employee has served the organization is greater than 3 then a bonus of Rs. 2500/- is given to the employee.

If the years of service are not greater than 3, then the program should do nothing.

```
main( )                /* Calculation of bonus */
{
    int bonus, cy, yoj, yr_of_ser ;
    printf("Enter current year and year of joining ");
    scanf( "%d %d", &cy, &yoy ) ;
    yr_of_ser = cy - yoj ;
    if ( yr_of_ser > 3 )
    {
        bonus = 2500 ;
        printf ( "Bonus = Rs. %d", bonus ) ;
    }
}
```

Exercise

In a company an employee is paid as under:

- If his basic salary is less than Rs. 1500, then HRA = 10% of basic salary and DA = 90% of basic salary.
- If his salary is either equal to or above Rs. 1500, then HRA = Rs. 500 and DA = 98% of basic salary.
- If the employee's salary is input through the keyboard write a program to find his gross salary.

program

```
/* Calculation of gross salary */
main( )
{
    float bs, gs, da, hra ;
    printf ( "Enter basic salary " );
    scanf ( "%f", &bs );
    if ( bs < 1500 )
    {
        hra = bs * 10 / 100 ;
        da = bs * 90 / 100 ;
    }
    else
    {
        hra = 500 ;
        da = bs * 98 / 100 ;
    }
    gs = bs + hra + da ;
    printf("gross salary= Rs. %f",gs);
}
```

exercise

The marks obtained by a student in 5 different subjects are input through the keyboard. The student gets a division as per the following rules:

- Percentage above or equal to 60 - First division
- Percentage between 50 and 59 - Second division
- Percentage between 40 and 49 - Third division
- Percentage less than 40 - Fail


```
/* Method – I (nested if)*/
```

```
main( )  
{  
    int m1, m2, m3, m4, m5, per ;  
    printf("Enter marks in 5 subjects") ;  
    scanf ("%d %d %d %d %d", &m1, &m2,  
        &m3, &m4, &m5 );  
    per =( m1+m2+m3+m4+m5)/5;
```

```
    if ( per >= 60 )  
        printf ( "First division " ) ;  
    else  
    {  
        if ( per >= 50 )  
            printf ( "Second division" ) ;  
        else  
        {  
            if ( per >= 40 )  
                printf ( "Third division" ) ;  
            else  
                printf ( "Fail" ) ;  
        }  
    }  
}
```

```
/* Method – II (simple if)*/
```

```
main( )  
{  
    int m1, m2, m3, m4, m5, per ;  
    printf("Enter marks in 5 subjects") ;  
    scanf ("%d %d %d %d %d", &m1, &m2,  
        &m3, &m4, &m5 );  
    per =( m1+m2+m3+m4+m5)/5;
```

```
    if ( per >= 60 )  
        printf ( "First division" ) ;  
    if((per>=50)&&(per<60))  
        printf ( "Second division" ) ;  
    if((per>=40)&&(per<50))  
        printf ( "Third division" ) ;  
    if ( per < 40 )  
        printf ( "Fail" ) ;  
}
```

```
/* Method – III else if ladder demo */
```

```
main( )
```

```
{
```

```
int m1, m2, m3, m4, m5, per ;
```

```
printf("Enter marks in 5 subjects") ;
```

```
scanf ("%d %d %d %d %d", &m1,  
    &m2, &m3, &m4, &m5 );
```

```
per =( m1+m2+m3+m4+m5)/5;
```

```
if ( per >= 60 )
```

```
    printf ( "First division" ) ;
```

```
else if ( per >= 50 )
```

```
    printf ( "Second division" ) ;
```

```
else if ( per >= 40 )
```

```
    printf ( "Third division" ) ;
```

```
else
```

```
    printf ( "fail" ) ;
```

```
}
```

The switch Statement

- When we need to select any one of the many alternatives we can use if statement
- But the complexity of such a program increases when the number of alternatives increases.
- The program became difficult to read and follow.
- To solve this problem C has a multiway decision statement known as **switch**

The switch Statement

- The *switch* statement **evaluates an expression**, then attempts to match the result to one of several possible *cases*
- Each case contains a value and a list of statements
- The flow of control transfers to statement associated with the first case value that matches

The switch Statement

- The general syntax of a `switch` statement is:

```
switch ( expression )
{
    case value1 :
        statement-list1;
        break;

    case value2 :
        statement-list2;
        break;

    case value3 :
        statement-list3;
        break;

    case ...

    default:
        statement-list;
        break;

}
```

Statement-x

Working of switch Statement

- The *switch* statement evaluates an expression(*Expression can be integer or character*)
- When switch is executed the *value of the expression is successfully compared against the values ,value1,value 2 ...*
- *When a match is found, a block of statements associated with the case is executed*
- Often a *break* statement is used as the last statement in each case's statement list

break statement

- A *break* statement signals the end of a particular case and cause an exit from switch statement and transfer the control to statement-x
- If a *break* statement is not used, the flow of control will continue into the next case.

default case

- A ***switch*** statement can have an optional *default case*
- If the value of the expression doesn't match any of the **case values** default statement is executed
- If there is no default case, and no other value matches, control falls through to the statement-x after the switch

The switch Statement

- The expression of a `switch` statement must result in an *integral type*, meaning an integer or a `char`
- It cannot be a floating point value (`float` or `double`)
- The implicit test condition in a `switch` statement is equality
- You cannot perform relational checks with a `switch` statement

```
#include <stdio.h>

int main()
{
    int x =2;
    switch (x)
    {
        case 1: printf("Choice is 1");
                break;
        case 2: printf("Choice is 2");
                break;
        case 3: printf("Choice is 3");
                break;
        default: printf("Choice other than 1, 2 and 3");
                break;
    }
    return 0;
}
```

Using switch

- The variables **x** and **y** refer to numbers. Write a **code segment that** prompts the user for an arithmetic operator and prints the value obtained by applying that operator to **x** and **y**.

```
#include <stdio.h>
void main()
{
    int x,y,sum,sub,mul;
    float div;
    char op;
    printf("enter operator");
    scanf("%c",&op);
    printf("enter x and y");
    scanf("%d%d",&x,&y);
    switch(op)
    {
        case '+':
            sum=x+y;
            printf("sum is %d",sum);
            break;

        case '-':
            sub=x-y;
            printf(" subtracted value  is %d",sub);
            break;

        case '*':
            mul=x*y;
            printf(" multiplied  value is %d",mul);
            break;

        case '/':
            div=x/y;
            printf(" divided value  is %f",div);
            break;

        default:
            printf("Invalid operator");

    }
}
```

Predict the output for input 3

```
void main()
{
    int day;
    printf("enter day");
    scanf("%d",&day);
    switch(day)
    {
        case 1:printf("monday");
                break;
        case 2:printf("tuesday");
                break;
        case 3:printf("wednesday\n");

        case 4:printf("thursday\n");

        case 5:printf("friday\n");

        default: printf("invalid number");
    }
}
```

Calculate grade using switch

```
main( )
{
    int m ,G;
    printf ("Enter mark in 100");
    scanf ( "%d", &m);
    Printf("\nGrade:");
    G=m/10;
    switch (G)
    {
        case 10 :
        case 9 :
        case 8:
            printf ( "Distinction" ) ;
            break ;
```

```
        case 7 :
        case 6 :
            printf ( "first class" ) ;
            break ;
        case 5 :
            printf ( "second class" ) ;
            break ;
        default :
            printf ( "failed" ) ;
            break;
    }
```

Output

```
Enter mark in 100
70
Grade: first class
```

Loop Control Statements

- To perform looping operations until the given condition is true.
- Control comes out of the loop statements once condition becomes false.

3 Types :

1. **for**
2. **while**
3. **do-while**

Loop Name	Syntax
for	<pre>for (exp1; exp2; expr3) { statements; }</pre> <p>Where, exp1 – variable initialization (Example: i=0, j=2, k=3) exp2 – condition checking (Example: i>5, j<3, k=3) exp3 – increment/decrement (Example: ++i, j–, ++k)</p>
while	<pre>while (condition) { statements; }</pre> <p>where, condition might be a>5, i<10</p>
do while	<pre>do { statements; } while (condition);</pre> <p>where, condition might be a>5, i<10</p>

while loop

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

while (test condition)

{

body of the loop

}

while loop working

- Entry controlled loop.
- *Test condition is evaluated and if it is true the body of the loop is executed*
- After execution the condition is again evaluated and if it is true the body of the loop is executed again
- The **process** is repeated as long as the loop repetition condition is **true**.
- A loop is called an **infinite loop** if the loop condition is true.
- When the condition becomes false the control is transferred out of the loop.

1 Print numbers up to n

```
#include<stdio.h>
void main()
{
    int limit, i=0;
    printf("enter the limit");
    scanf("%d",&limit);
    while(i<limit)
    {
        printf("%d",i);
        i=i+1;
    }
}
```

```
main( )  
{  
    int i = 5 ;  
    while(i >=1)  
    {  
        printf("\nhello");  
        i=i-1;  
    }  
}
```

Instead of incrementing a loop counter, we can even decrement manage to get the body of the loop executed repeatedly.

```
main( )  
{  
    float a = 10.0 ;  
    while (a <= 10.5 )  
    {  
        printf("hai");  
        a = a + 0.1 ;  
    }  
}
```

It is not necessary that a loop counter must only be an int. It can be even float

Even floating point loop counters can be decremented. Once again the increment and decrement could be by any value, not necessarily 1.

Factorial of a number using while

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

```
void main()
```

```
{  int limit,i=1,f=1;
```

```
    printf("enter the limit");
```

```
    scanf("%d",&limit);
```

```
    while(i<=limit)
```

```
    {  f=f*i;
```

```
        i=i+1;
```

```
    }
```

```
        printf("factorial is %d",f);
```

```
}
```

display the consecutive digits 0, 1, 2, . . . ,9,

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

```
main( )
```

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


- The condition being tested may use relational or logical operators as shown in the following examples:
- `while(i<=10)`
- `while(i >=10&& j<=15)`
- `while(j>10&&(b<15 || c<20))`

The program to check no is prime or not

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

```
#include<math.h>
```

```
void main()
```

```
{  
    int n, i=2, flag=0;  
    printf("enter the numbers");  
    scanf("%d",&n);  
    if(n==1)  
        printf("number is neither prime  
nor composite");  
    else  
        while(i<=sqrt(n))  
        {  
            if(n%i==0)  
                flag=1;  
            i=i+1;  
        }  
}
```

```
if(flag==1)
```

```
printf("not prime");
```

```
else
```

```
printf("prime");
```

```
}
```

Find the sum of n natural numbers

```
#include<stdio.h>
main()
{
    int n,s=0;
    printf("Enter the limit");
    scanf("%d",&n);
    int i=0;
    while(i<=n)
    {
        s=s+i;
        i++;
    }
    printf("sum = %d",s);
}
```

The do-while Statement in C

- Exit controlled loop
- The syntax of do-while statement in C:

do

{

statements;

} while (test condition);

The do-while Statement in C

- The *statement* is first executed.
- If the **loop repetition condition** is true, the *statement* is repeated.
- Otherwise, the loop is exited.

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

```
void main()
```

```
{ int i=5;
```

```
    do
```

```
        {printf("hello");
```

output

hello

```
        i=i+1;
```

```
    }
```

```
while(i<5);
```

```
}
```

Factorial of a number using **do while**

```
#include<stdio.h>
void main()
{
    int limit,i=1,f=1;
    printf("enter the limit");
    scanf("%d",&limit);
    do
    {
        f=f*i;
        i=i+1;
    }while(i<=limit);
    printf("factorial is %d",f);
}
```

For loop syntax

```
for (initialization; test condition; increment or decrement)
{
    C statements needs to be repeated
}
```


For loop syntax

Step 1: first initialization happens and the counter variable gets initialized, here variable is `i`, which has been assigned by value 1.

Step 2: then condition checks happen, where variable has been tested for a given condition, if the condition results in true then C statements enclosed in loop body gets executed by compiler, otherwise control skips the loop and continue with the next statement following loop.

Step 3: After successful execution of loop's body, the counter variable is incremented or decremented, depending on the operation (`++` or `--`).

Program to print first n numbers using for loop

```
#include<stdio.h>
void main()
{
    int limit,i;
    printf("enter the limit");
    scanf("%d",&limit);
    for(i=1;i<=limit;i=i+1)
    {
        printf("%d",i);

    }
}
```

Factorial of a number using for loop

```
#include<stdio.h>
void main()
{
    int limit,f=1,i;
    printf("enter the limit");
    scanf("%d",&limit);
    for(i=1;i<=limit;i++)
    {
        f=f*i;

    }
    printf("factorial is %d",f);
}
```

```
/* display the numbers 0  
    through 9 */
```

```
#include <stdio.h>  
  
void main()  
{  
  
    int digit=0;  
    for( ;digit<=9; )  
        printf("%d\n",digit++);  
}
```

Here is still another example of a C program that generates the consecutive integers 0, 1, 2, . . . ,9, with one digit on each line. We now use a for statement in which two of the three expressions are omitted.

From a syntactic standpoint **all three expressions need not be included in the for statement, though the semicolons must be present.**

However, the consequences of an omission should be clearly understood.

Reverse a number using while loop

```
#include<stdio.h>

void main()
{
    int num,rev=0,digit;
    printf("enter number");
    scanf("%d",&num);
    while(num!=0)
    {
        digit=num%10;
        rev=rev*10+digit;
        num=num/10;

    }
    printf("reverse id %d",rev);
}
```

C program to reverse a number using for loop

```
#include<stdio.h>

void main()
{
    int num,rev=0,digit;
    printf("enter number");
    scanf("%d",&num);
    for(;num!=0;num=num/10)
    {
        digit=num%10;
        rev=rev*10+digit;
    }
    printf("reverse id %d",rev);
}
```

Palindrome checking using while

```
#include<stdio.h>
void main()
{
    int num,rev=0,digit,n;
    printf("enter number");
    scanf("%d",&num);
    n=num;
    while(num!=0)
    {
        digit=num%10;
        num=num/10;
        rev=rev*10+digit;
    }
    if(n==rev)
        printf("number is palindrome");
    else
        printf("number is not palindrome");
}
```

Palindrome checking

```
#include<stdio.h>
void main()
{
    int num,rev=0,digit,n;
    printf("enter number");
    scanf("%d",&num);
    n=num;
    for(;num!=0;num=num/10)
    {

        digit=num%10;

        rev=rev*10+digit;

    }
    if(n==rev)
        printf("number is palindrome");
    else
        printf("number is not palindrome");
}
```


check the given number is Armstrong or not

```
#include<stdio.h>
void main()
{
    int num,arm=0,digit,n;
    printf("enter number");
    scanf("%d",&num);
    n=num;
    for(;num!=0;num=num/10)
    {
        digit=num%10;
        arm=arm+digit*digit*digit;
    }
    if(n==arm)
        printf("number is armstrong");
    else
        printf("number is not armstrong");
}
```

```
#include<stdio.h>
main()
{
    int s,e,sum,i,n,d;
    printf("enter the range\n");
    scanf("%d%d\n\n",&s,&e);
    printf("amstrong nos are\n");
    for(i=s;i<=e;i++)
    {
        sum=0;
        n=i;
        while(n!=0)
        {
            d=n%10;
            sum=sum+(d*d*d);
            n=n/10;
        }
    }
}
```

**generate Armstrong
number
with in a range**

```
if(sum==i)
{
    printf("%d\n",i);
}
if(sum!=i)
{
    printf("no amstrong  
no\n");
}
}
```

Nested Loops

- The way if statements can be nested, similarly whiles and for can also be nested
- Nested loops consist of an **outer loop** with one or more **inner loops**.

- e.g.,

```
for (i=1;i<=100;i++){
```

```
    for(j=1;j<=50;j++){
```

```
        ...
```

```
    }
```

```
}
```

Outer loop

Inner loop

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

```
#include <stdio.h>

int main()
{ for (int i=0; i<2; i++)
  { for (int j=0; j<4; j++)
    { printf("%d, %d\n",i ,j);
      }
    }
  return 0;
}
```

Output:

0, 0

0, 1

0, 2

0, 3

1, 0

1, 1

1, 2

1, 3

Program to print pattern

*

**

```
#include <stdio.h>

int main()
{ int rows, i, j;
  printf("Enter number of rows\n");
  scanf("%d", &rows);
  for (i = 1; i <= rows; i++)
  {
    for(j = 1; j <= i; j++)
    {
      printf("*");
    }
    printf("\n");
  }
  return 0;
}
```

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

```
#include <stdio.h>
int main()
{
    int i, j, rows;
    printf("Enter number of rows: ");
    scanf("%d",&rows);

    for(i=1; i<=rows; ++i)
    {
        for(j=1; j<=i; ++j)
        {
            printf("%d ",j);
        }
        printf("\n");
    }
    return 0;
}
```


Floyd's triangle

Enter the number of rows of Floyd's triangle to print

```
7
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
16 17 18 19 20 21
22 23 24 25 26 27 28
```

```
int main()
{
    int n, i, c, a = 1;
    printf("Enter the number of rows of Floyd's triangle to print\n");
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        for (c = 1; c <= i; c++)
        {
            printf("%d ",a);
            ++a;
        }
        printf("\n");
    }
}
```

Floyd's triangle

Multiple Initialisations in the *for Loop*

- The initialisation expression of the **for loop** can **contain more than one statement separated by a comma.**
- `for (i = 1, j = 2 ; j <= 10 ; j++)`
- Multiple statements can also be used in the incrementation expression of **for loop**; **i.e., you can increment (or decrement) two or more variables at the same time. However, only one expression is allowed in the test expression.**

*

* * *

* * * * *

* * * * * *

* * * * * * * *

```
#include <stdio.h>
int main()
{
    int row, c, n, i, s;

    printf("Enter the number of rows in pyramid of stars you wish to see\n");
    scanf("%d", &n);
    s = n;
    for (row = 1; row <= n; row++) // Loop to print rows
    {
        for (c = 1; c<s; c++) // Loop to print spaces in a row

            { printf(" ");}
        s--;
        for (i = 1; i <= 2*row - 1; i++) // Loop to print stars in a row
            {printf("*");}
        printf("\n");
    }
    return 0;}
```

Case Control Statements

The statements which are used to execute only specific block of statements in a series of blocks are called case control statements.

1. break
2. continue
3. goto

break statement

- λ Break statement is used to terminate the while loops, switch case loops and for loops from the subsequent execution.

break

```
#include <stdio.h>

int main()
{
    for(int i=0;i<5;i++)
    {   if(i==2)
        {   printf("\nComing out at%d",i);
            break;}
        printf("%d ",i);
    }
}
```

output
0 1

Coming out at 2

Continue statement

- λ used to continue the next iteration of for loop, while loop and do-while loops.

Example

```
#include <stdio.h>
int main()
{
    int i;
    for(i=0;i<6;i++)
    {   if(i==4)
        {
            printf("\nComing out at%d",i);
            continue;
        }
        printf("\n %d ",i);
    }
}
```

goto

statements

- used to transfer the normal flow of a program to the specified label in the program.

Syntax :

```
{  
    .....  
    go to LABEL;  
    .....  
    .....  
    LABEL:  
    statements;  
}
```

The *goto* Keyword

- where *label* is an identifier that is used to label the target statement to which control will be transferred.
- Label can be anywhere in the program ie before or after go to label
- 2 types of go to statements
 - *Forward jump*
 - *Backward jump*
- *Forward jump*
 - The label is placed *AFTER* goto label.
 - Some statements are skipped
- *Backward jump*
 - The label is placed *BEFORE* goto label.
 - Loop will be formed

```
#include<stdio.h>
int main()
{
    printf ("Hello World");
    goto Label;
    printf("How are you");
    printf("Are you Okey");
    Label:
    printf("Hope you are fine");
}
```

Forward jump in go to

```
#include <stdio.h>
main()
{for(int i=0;i<5;i++)
{
if(i==2)
{
printf("\nWe are using goto statement when i = 2");
goto HAI;
}
printf("%d ",i);
}
HAI : printf("\nhello\n");
}
```

Output

0 1

We are using goto statement
when i = 2
hello

Backward jump in go to

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

```
int main()
```

```
{
```

```
    int a=1;
```

```
    Read:
```

```
    printf ("HELLO\n");
```

```
    a=a+1;
```

```
    if(a<=5)
```

```
        goto Read;
```

```
}
```