CONDITIONAL AND LOOP STATEMENTS IN C

MOHD ADIL

ASSISTANT PROFESSOR

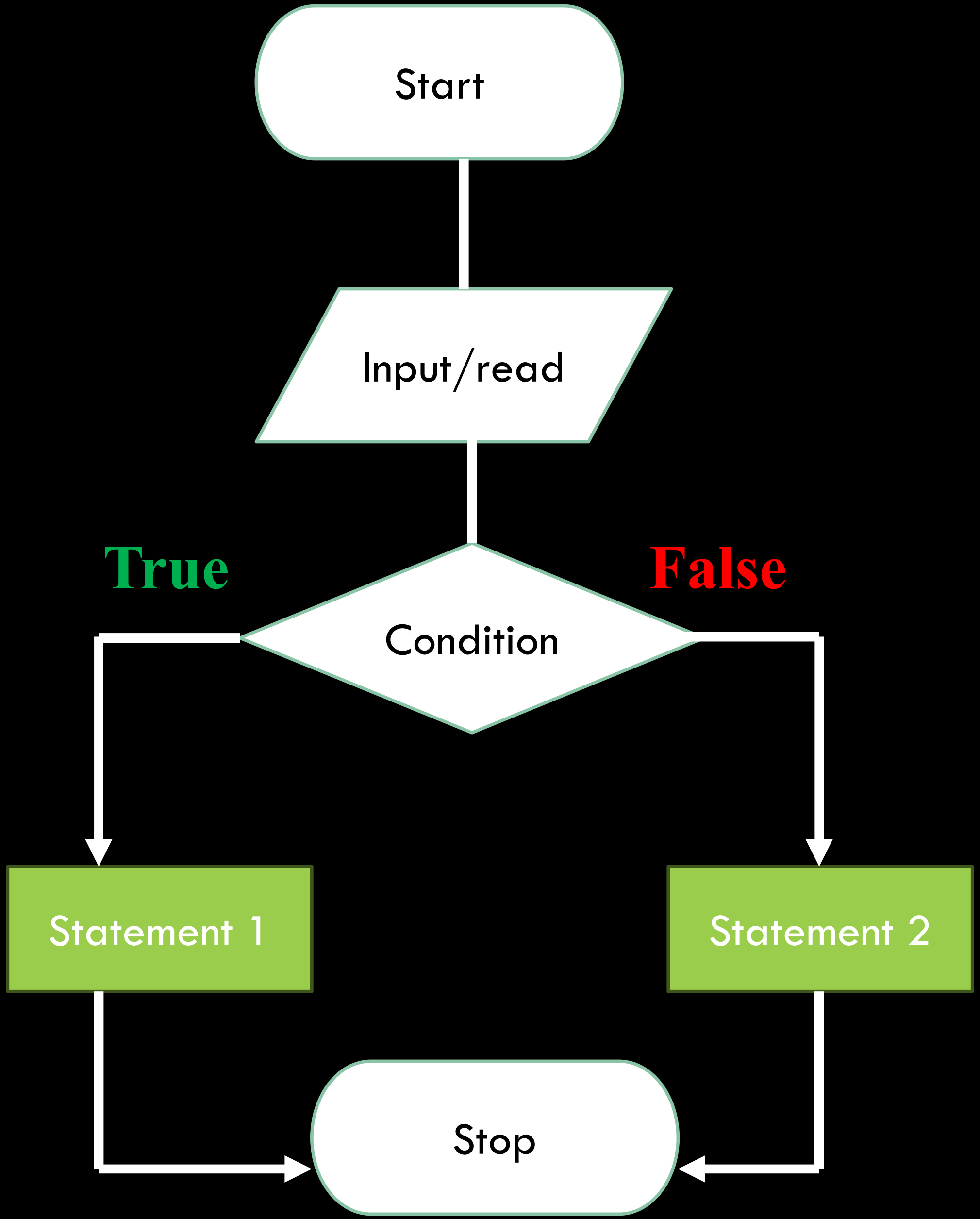
THE IF-ELSE STATEMENT

- The if block executes if the condition is true.
- The else block executes if the condition is false
- Syntax:

```
if(condition)
   statement1;
  else
      statement2;
```

THE IF-ELSE STATEMENT

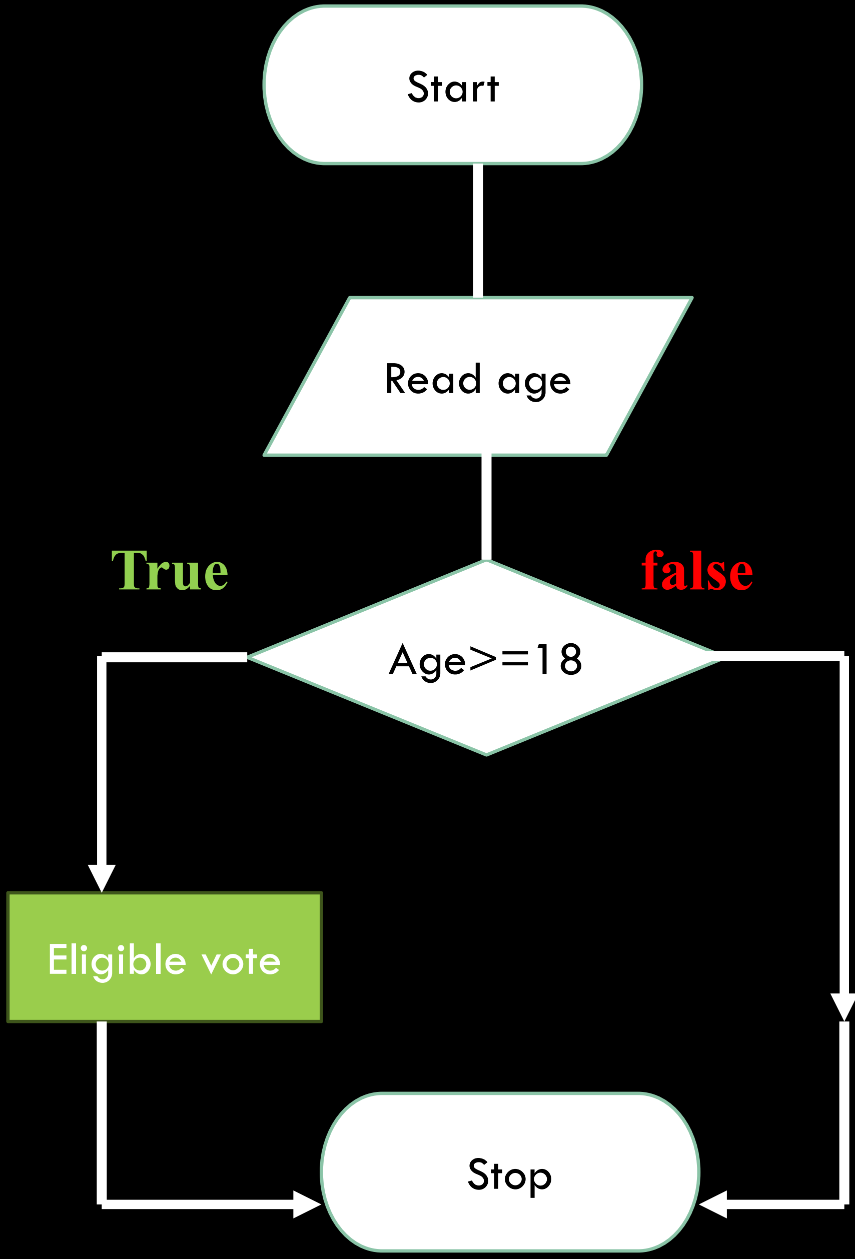
- **Flow chart:**



THE IF-ELSE STATEMENT

- **Program**

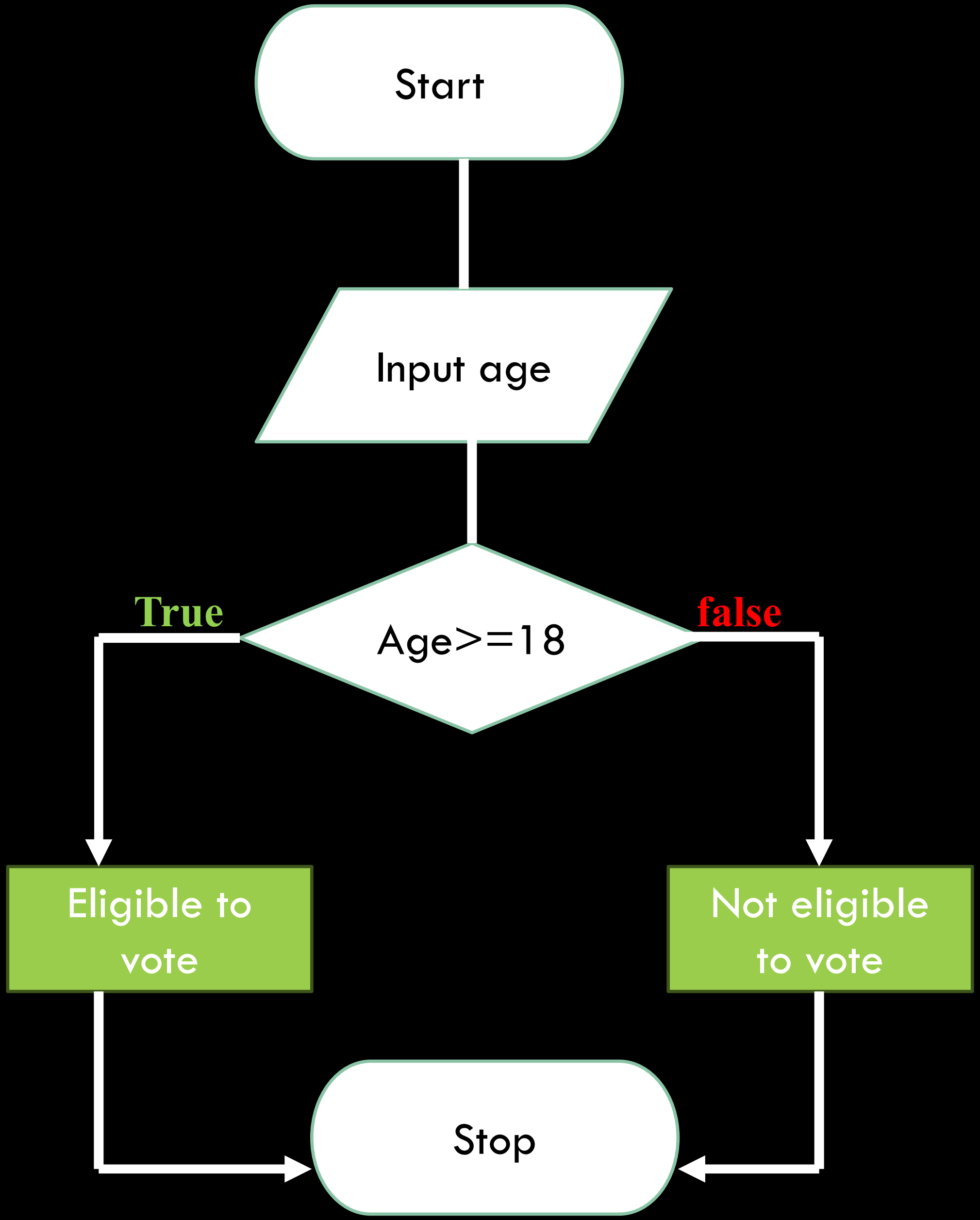
```
#include <stdio.h>
int main (){
int age=15;
if (age>=18)
            printf("Eligible to vote.");
return 0;
}
```



THE IF-ELSE STATEMENT

- **Flow chart:**
- **Program**

```
#include <stdio.h>
int main (){
int age=15;
if (age>=18)
  printf(“Eligible to vote.”);
 else
   printf(“you are not eligible to vote”
return 0;
```



MULTIPLE STATEMENTS WITHIN IF-ELSE STATEMENT

- The **default scope** of the **if** and **else** statement is the **next statement.**
- **Multiple statements** need to be **enclosed** within **curly braces.**

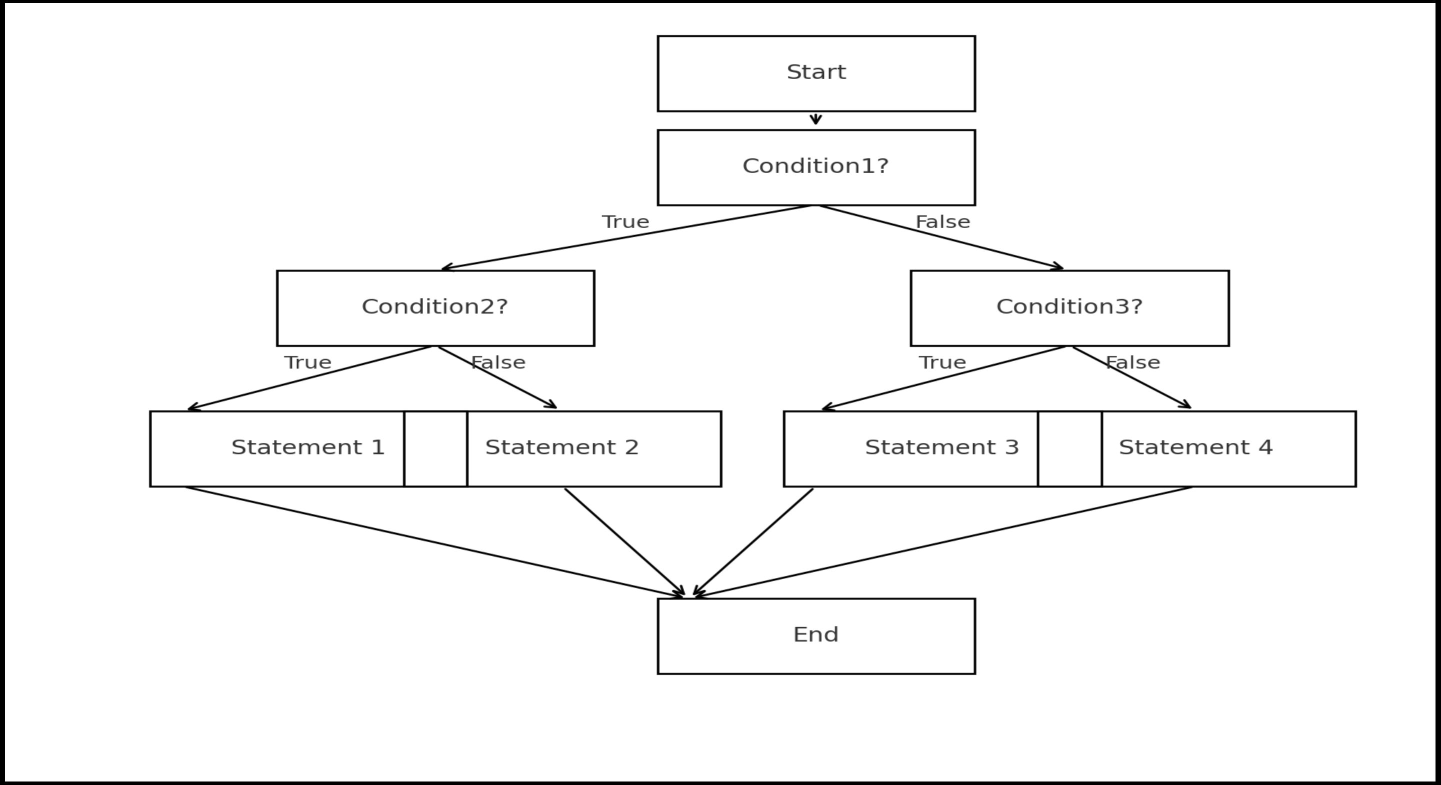
```
#include<stdio.h>
Int main(){
Int age =30;
If(age>=18){
printf("Eligible to vote.");
printf("Eligible for driving licence.");
}
else{
printf("Eligible to vote.");
printf("Eligible for driving licence");
}
```

THE NESTED IF-ELSE STATEMENT

- It refers to an if or else statement that contains another if-else statement.
- Syntax:

```
if (Condition1){
          if (Condition2){
             Statement 1;
           }else {
                statement 2;
           } else{
              if (Condition3){
                statement 3;
              } else {
                 statement4;
                }
              }
```

Flow chart:



THE NESTED IF-ELSE STATEMENT

Example:

```
#include<stdio.h>
Int main(){
Int age =30;
If(age>=18){
printf(“Eligible to vote.”);
printf(“Eligible for driving licence.”);
}
else{
   printf(“Not Eligible to vote.”);
   if(age>=16){
     printf(“Eligible for a learner’s permit but not a driving license.\n”);
  }
  else{
   printf(“Not eligible for a learner’s permit or driving license.\n”);
}
return 0;
}
```

This inner **if-else** block can be placed inside the **if** part, inside the **else** part, or even in both. The whole structure together is called a **nested if-else.**

N- IF-ELSE STATEMENT

Problem statement:

Write a C program that checks whether a person is eligible to apply for a driving license. The person must meet the following criteria:

- **Age must be at least 18.**
- **Must have passed an eyesight test.**
- **Must have passed a traffic rules knowledge test.**

NESTED IF-ELSE STATEMENT

Program

```
#include <stdio.h>
int main(){
int age =18, ETPassed=1, TTPassed=1;
//check the eligibility using Nested if-else
If(age>=18){
    if(ETPassed==1){
        if(TTPassed==1){
                printf(“You are eligible to apply for a driving license.\n”);
}
 else {
                printf(“You are not  eligible to apply for a driving license.\n”);
        }
    } else {
            printf(“You are not eligible to apply for a driving license.\n”);
     }
}  else {
     printf(“You are not eligible to apply for a driving license.\n”);
}
return 0;
}
```

IF-ELSE STATEMENT AND LOGICAL OPERATOR

- **Program**
- **#include <stdio.h>**
- **int main(){**
- **int age =18, ETPassed=1, TTPassed=1;**
- **//check the eligibility using Logical operators**
- **If(age>=18 && ETPassed==1 && TTPassed==1){**
- **printf(“You are eligible to apply for a driving license.\n”);**
- **} else {**
- **printf(“You are not eligible to apply for a driving license.\n”);**
- **}**
- **return 0;**
- **}**

Here, we combined all the conditions into one if statement using the logical AND operator (&&). If all three conditions are satisfied, then the *if block* will execute and the program will print: *“You are eligible to apply for a driving license.”* Otherwise, the *else block* will execute.

THE ELSE IF STATEMENT

```
Program
#include <stdio.h>
int main(){
int age =16, ETPassed=0, TTPassed=0;
Int isEligible=1;
If(age<18) {
   print(“You are not eligible: Age must be at least 18.\n”);
   isEligible=0;
}
If(ETPassed!=1){
print(“You are not eligible: Eye test is failed\n”);
isEligible=0;
}
if(TTPassed!=1){
print(“You are not eligible: Technical test is failed\n”);
isEligible=0;
}
If (isEligible ==1){
print(“You are eligible: to apply for a driving license.\\n”);
}
return 0;
```

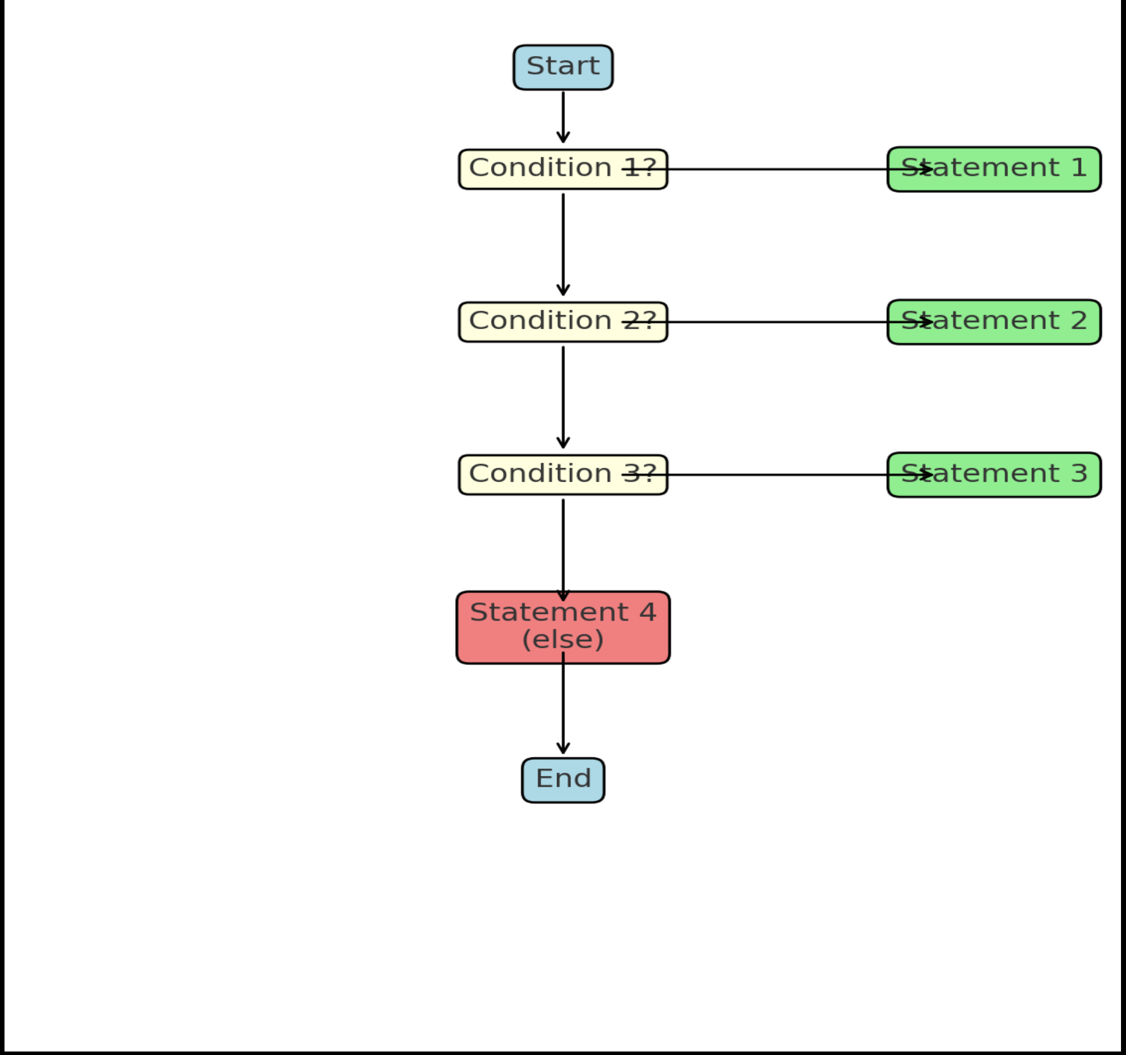
We are checking each condition separately.
The advantage is that the program clearly tells the **reason** for not being eligible.
If more than one condition fails, it will print multiple reasons."

THE ELSE IF STATEMENT

Program

```
#include <stdio.h>
int main(){
int age =16, ETPassed=0, TTPassed=0;
Int isEligible=1;
If(age<18) {
    print(“You are not eligible: Age must be at least 18.\n”);
    isEligible=0;
}
else if(ETPassed!=1){
print(“You are not eligible: Eye test is failed\n”);
isEligible=0;
}
}
else if(TTPassed!=1){
print(“You are not eligible: Technical test is failed\n”);
isEligible=0;
}
else{
print(“You are not eligible: to apply for a driving license.\\n”);
}
return 0;
```

THE ELSE IF STATEMENT



In this program, we used **else if** instead of writing only separate if statements.

The difference is:

When the first if condition is **true**, its block executes and the program skips all the remaining conditions.

If the first if is **false**, then the program checks the next else if.

Again, if that is false, it checks the next one.

Finally, if none of the conditions are true, the else block executes.

This way, only **one block** will execute, and the remaining conditions will not even be checked once a match is found.

THE SWITCH STATEMENT

- It allows executing a set of statements based on an expression.

```
Syntax  switch (integral expression)
      {
              case integral constant:
                     statement1;
               case integral constant:
                     statement2;
                case integral constant:
                     statement3;
                 default:
                      statement 4;
}
```

With the help of a switch statement, we can execute a set of statements based on the result of an expression. We pass an expression to the switch, and the switch will select and execute one set of statements depending on the value of that expression.

THE SWITCH STATEMENT

- It allows executing a set of statements based on an expression.
  - Example

```
#include <stdio.h>
int main (){
int var=1
 switch (var)
    {
          case 1:
                printf(“inside case 1.\n”)
           case 2:
                printf(“inside case 2.\n”);
          case 3:
                printf(“inside case 3.\n”)

           default:
                printf(“inside
default.\n”)
}
}
```

THE SWITCH STATEMENT -PROPERTIES

1. **Case Labels with integral Types:**

No data type other than integral type is allowed as a case label

- Invalid code

```
Switch (a){
Case 4.5:
Statement;
Case 3.14159:
Statement;
Default:
Statement
}
```

- valid code

```
Switch (a){
Case 4:
Statement;
Case 3:
Statement;
Default:
Statement
}
```

THE SWITCH STATEMENT -PROPERTIES

2. Character-Based Cases: Character literals can be used as case labels as internally they are treated as integers.

```
Switch (letter){
Case ‘A’:
printf(“Grade A.\n”);
break;
Case ‘B’:
printf(“Grade A.\n”);
break;
Default:
printf(“unknown grade \n”);
}
```

THE SWITCH STATEMENT -PROPERTIES

3. Grouped Case Labels: Instead of adding same code under each case, multiple cases can be grouped

```
Switch (a){
Case ‘A’:
Case ‘E’:
Case ‘I’:
Case ‘O’:
Case ‘U’;
printf(“vowel \n”);
Default:
printf(“consonant \n”);
}
```

THE SWITCH STATEMENT -PROPERTIES

4. Default case Placement

   the default case can appear anywhere in the switch case

```
Switch (code){
default:
printf("unknown \n");
break;
Case 1:
printf("code1 \n");
Break;
Case 2:
printf("code2 \n");
}
```

THE SWITCH STATEMENT -PROPERTIES

5. Non- Constant Case Labels: Variable as case labels are not allowed

```
Switch (option){
Case 1:
Printf("Option 1.\n") ;
break;
Case userInput:
Printf("user option\n");
Break;
Default:
Statement
}
```

THE CONDITIONAL OPERATOR

- It’s a ternary operator
- Represented by ? :
- Shorthand way of writing if-else statement.
- Syntax condition ? Expression_if_true : expression_if_false

```
#include <stdio.h>

Int main(){

Int a=5,b=10;

Int max=(a>b)? a : b;

Printf(“the maximum value is
%d\n”,max);

Return 0;

}
```

THE NESTED CONDITIONAL OPERATOR

- Sometimes, we need to check **more than one condition** using the ternary operator.
- In that case, we can **nest** one conditional operator inside another.
- This is called the **nested conditional operator.**

syntax

```
condition1  ? expression1 :
   (condition2 ? expression2 :
      (condition3 ? expression3 :
expression4));
```

THE NESTED CONDITIONAL OPERATOR

- First, condition1 is checked
- If true => expression1 is executed
- If false → second condition expression2 is checked
- This process continues until one condition becomes true.
- If none are true, the **last expression** executes.

syntax

```
condition1 ? expression1 :
  (condition2 ? expression2 :
    (condition3 ? expression3 :
expression4));
```

THE NESTED CONDITIONAL OPERATOR

- Example Find the largest of three numbers.

```
#include <stdio.h>
int main() {
    int a = 10, b = 20, c = 15;
    int largest = (a > b) ?
                ((a > c) ? a : c) :
                ((b > c) ? b : c);
    printf("The largest number is %d\n", largest);
    return 0;
}
```

Output

```
The largest number is 20
```

- **Key Points:**
- Nested conditional operator is a **short-hand replacement** for multiple if-else statements.
- It makes code **compact** but sometimes **less readable** if overused.

BREAK KEYWORD

- **The break** keyword is used to **terminate** the execution of a loop or a switch statement immediately.
- When break is encountered, the control **jumps out** of the current block.

break in switch case:

- In a switch statement, each case is followed by some statements.
- Without break once a case is matched, all the statements of the following cases will also be executed (this is called **fall-through**).

Switch without break

```
#include <stdio.h>

int main() {

    int day = 2;

 switch(day) {

        case 1: printf("Monday\n");

        case 2: printf("Tuesday\n");

        case 3: printf("Wednesday\n");

        default: printf("Other day\n");

    }

    return 0;

}
```

Output

```
Tuesday
Wednesday
Other day
```

BREAK KEYWORD

- **The break** keyword is used to **terminate** the execution of a loop or a switch statement immediately.
- When break is encountered, the control **jumps out** of the current block.

break in switch case:

- In a switch statement, each case is followed by some statements.
- Without break once a case is matched, all the statements of the following cases will also be executed (this is called **fall-through**).

Switch with break

```
#include <stdio.h>

#include <stdio.h>

int main() {

    int day = 2;

    switch(day) {

        case 1: printf("Monday\n"); break;

        case 2: printf("Tuesday\n"); break;

        case 3: printf("Wednesday\n"); break;

        default: printf("Other day\n");

    }

    return 0;

}
```

Output

```
Tuesday
```

Here, only the matched case executed because we used break

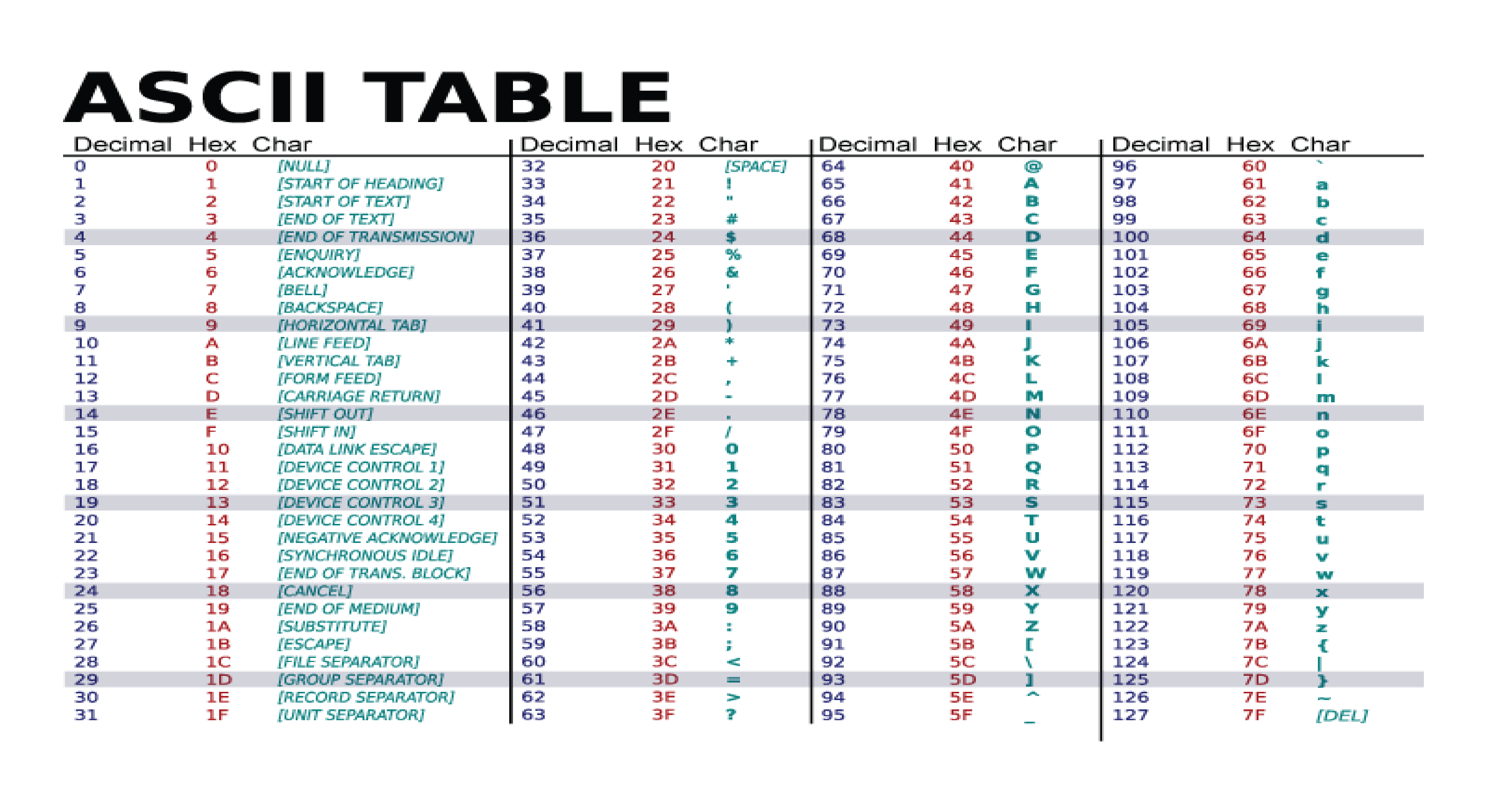
ASCII

- **ASCII** stands for **American Standard Code for Information Interchange.**
  It is a **character encoding.**
- Each character is represented by a number (code).
- **types of characters in ASCII: standard**
- **Control characters** (0–31): Non-printable, used for control (like newline, tab)
- **Printable characters** (32–126): Letters, digits, symbols, punctuation.

Why ASCII?

- Computers store everything in Binary (0’s and 1’s).
- ASCII provides a mapping from characters → numbers → binary, so text can be stored and processed easily.

ASCII



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]