



**SRI AKILANDESWARI WOMEN'S COLLEGE, WANDIWASH**

**PROGRAMMING IN C**  
Class : UG Computer Science

**Ms. P. GUNAVATHI**  
Assistant Professor  
Department of Computer Science

**SWAMY ABEDHANADHA EDUCATIONAL TRUST, WANDIWASH**

# MANAGING INPUT AND OUTPUT OPERATIONS

## Header Files

 Program that uses standard input output function must contain the statement `#include <stdio.h>`

 The file name `stdio.h` is an abbreviation for standard input – output header file.

## Reading and writing a Character:

 Input operation is reading a character from the standard input unit (keyboard).

 Output operation is writing it to the standard output unit(Screen).

# MANAGING INPUT AND OUTPUT OPERATIONS

## Reading a Character:

 It is used to accept a character in a C program.

 Done by using the function **getchar**.

## getchar():

When getchar() function will be encountered by C compiler while executing a program, the program will wait for the user to press a key from the keyboard.

## Syntax:

*Variable\_name = getchar();*

Variable name  Valid C name. Declared as char type.


## Example:

```
char name;
```

```
name = getchar();
```

# MANAGING INPUT AND OUTPUT OPERATIONS

## Writing a Character:

 It is used to accept a character in a C program.

 Done by using the function **putchar**.

## putchar():

 The function putchar() writes a single character, one at a time to the standard output device.

 When this statement is executed, the stored character will be displayed on the monitor.

## Syntax:

*putchar(Variable\_name );*

Variable name  $\longrightarrow$  Type char variable containing a character.

## Example:

```
answer = 'y';
```

```
putchar (answer);
```

# MANAGING INPUT AND OUTPUT OPERATIONS

## Reading a Character:

Example program:

```
#include<stdio.h>
int main()
{
char ch;
printf(“Enter a character”);
ch=getch();
printf(“The entered character is:”);
putchar(ch);
return 0;
}
```

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Input


When formatted input is required:

- When need to input numerical data which may required in calculations.
- When enter key itself is a part of the data.
- When need to input data in a particular format.
- The scanf() function is used to input data in a formatted manner.

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Input

 **scanf()** function is used to give data to the variable using keyboard.

 Used to input data in a formatted manner.

Syntax:

*scanf("control string", &variable1, &variable2....&variable n);*

In C to represent an address of any location an ampersand (&) is used.

✓ Control string  $\longrightarrow$  Specifies the field format in which the vales of variable are to be stored. Each format must be preceded by %

✓ Variable  $\longrightarrow$  Specify the address of location where the data is stored.

Variables separated by commas.

 Control string also known as **format string**.

 Control string contains field specifications.

# MANAGING INPUT AND OUTPUT OPERATIONS


## Formatted Input

It may include:

 Field (or format) specifications, consisting of

- The conversion character %,
- a data type character (or type specifier)
- An optional number specifying the field width.

 Blanks, tabs, or newlines.

 The data type character indicates the type of data that is to be assigned to the variable.

 The field width specifier is optional.



# MANAGING INPUT AND OUTPUT OPERATIONS

Formatted Input

Format specifiers.

1	%d, %i	Signed decimal integer
2	%x, %X	Unsigned hexadecimal integer (without leading 0x)
3	%o	Unsigned octal integer (without leading 0)
4	%u	Unsigned decimal integer
5	%c	Single character
6	%s	String
7	%f	Real number in decimal notation
8	%e, %E	Real number in exponential notation
9	%g, %G	Real number either f-type or e-type depending on the length of the value without insignificant zero
10	%%	%

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Input

### Formatting integer numbers:

The field specification for reading an integer number is: `%wd`

- The `%` sign → A conversion specification follows.
- `w` → Specifies the field width of the number to be read.
- `d` → Specifies data type, indicates that the number to be read is in integer mode.

### Example:

```
scanf("%2d, &num1);
```

This statement is used to read an integer data of width 2.

Input: 45678

`num1` will be assigned 45 (because of `%2d`).

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Input

Formatting integer numbers:

Example:

```
scanf("%d", &num1);
```

This statement is used to reads an integer data and assigns to variable num1.

Input: 45678

num1 will be assigned 45678

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Input

### Inputting Real numbers:

Example:

```
scanf(“%f”, &num1);
```

This statement is used to reads a floating point data and assigns to variable num1.

Input: 45.678

num1 will be assigned 45.678

### Inputting Character Strings:

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Input

Inputting character strings:

`%ws` or `%wc`

Example:

```
scanf("%4s", &name);
```

This statement is used to reads a string of data and assigns to variable name.

Input: good

name will be assigned good.

Example:

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

This statement is used to reads a single character of data and assigns to variable name.

Input: a

Name will be assigned a.

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Input

### Reading Mixed data type:

Use one scanf() statement to input a data line containing mixed mode data.

### Example:

```
scanf(“%d %c %f %s”, &count, &code, &ratio, &name);
```

Input: 20 a 5.46 world

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Input

Example Program:

```
#include<stdio.h>

int main()
{
int num;
float value;
char ch;
char name[8];
printf("Enter the values");
scanf("%d %f %c %s", &num , &value, &ch, &name);
printf("Entered %d and %f and %c and %s", num, value, ch, name);
return 0;
}
```

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Input

Example Program:

Input : 55 78.656 a computer

Output: Entered 55 and 78.656 and a and computer



# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Output

printf() statement is used to display the result on screen.

Syntax:

*printf(“control string”, variable1, variable2....variable n);*

Control String consists of three types of items:

- Characters that will be printed on the screen as they appear.
- Format specifications that define the output format for display of each item.
- Escape sequence characters such as \n, \t and \b.

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Output

Example Program:

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

```
Void main()
```

```
{
```

```
int num; /*Declaration*/
```

```
num = 10; /*Initialization*/ (Compile time initialization)
```

```
printf(“%d”, num);
```

```
}
```

# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Output

Example Program:

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

```
Void main()
```

```
{
```

```
int num; /*Declaration*/
```

```
scanf(“%d”, &num); /*Initialization*/ (Run time initialization)
```

```
printf(“%d”, num);
```

```
}
```






# MANAGING INPUT AND OUTPUT OPERATIONS

## Formatted Output

type	meaning	example
d, I	integer	<code>printf("%d", 10); /* prints 10 */</code>
x, X	integer(hex)	<code>printf("%x", 10); /* prints 0xa */</code>
U	unsigned integer	<code>printf("%u", 10); /* prints 10 */</code>
c	character	<code>printf("%c", 'A'); /* prints A */</code>
S	string	<code>printf("%s", "hello"); /* prints hello */</code>
f	float	<code>printf("%f", 2.3); /* prints 2.3 */</code>
d	double	<code>printf("%d", 2.3); /* prints 2.3 */</code>
e, E	float(exp)	1e3, 1.2E3, 1E-3
%	literal %	<code>printf("%d" %%, 10); /* prints 10% */</code>

# FLOW CHARTS

## Flowchart

-  Flowchart is a graphical representation of an algorithm.
-  Flowcharts use special shapes to represent different types of actions or steps in a process.
-  Programmers often use it as a program-planning tool to solve a problem.
-  It makes use of symbols which are connected among them to indicate the flow of information and processing.
-  Lines and arrows shows the sequence of the steps, and the relationships among them.

# FLOW CHARTS

## Flowchart Symbols

There are 6 basic symbols commonly used in flowcharting:

1. Terminal
2. Process
3. input/output
4. Decision
5. Connector
6. Predefined Process

### Common Flowchart symbols:

Rectangle Shape – Represents a process








Oval Shape – Represents the start and end

Diamond Shape – Represents a decision

Parallelogram – Represents input/output

# FLOW CHARTS

## Flowchart

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

# DECISION MAKING AND BRANCHING

Decision making statement:

The decision making statements are:

- Simple if statement
- If....else statement
- Nested if...else statement
- Else if ladder
- Switch statement
- Conditional operator statement
- Goto statement

These statements are known as decision-making statements.

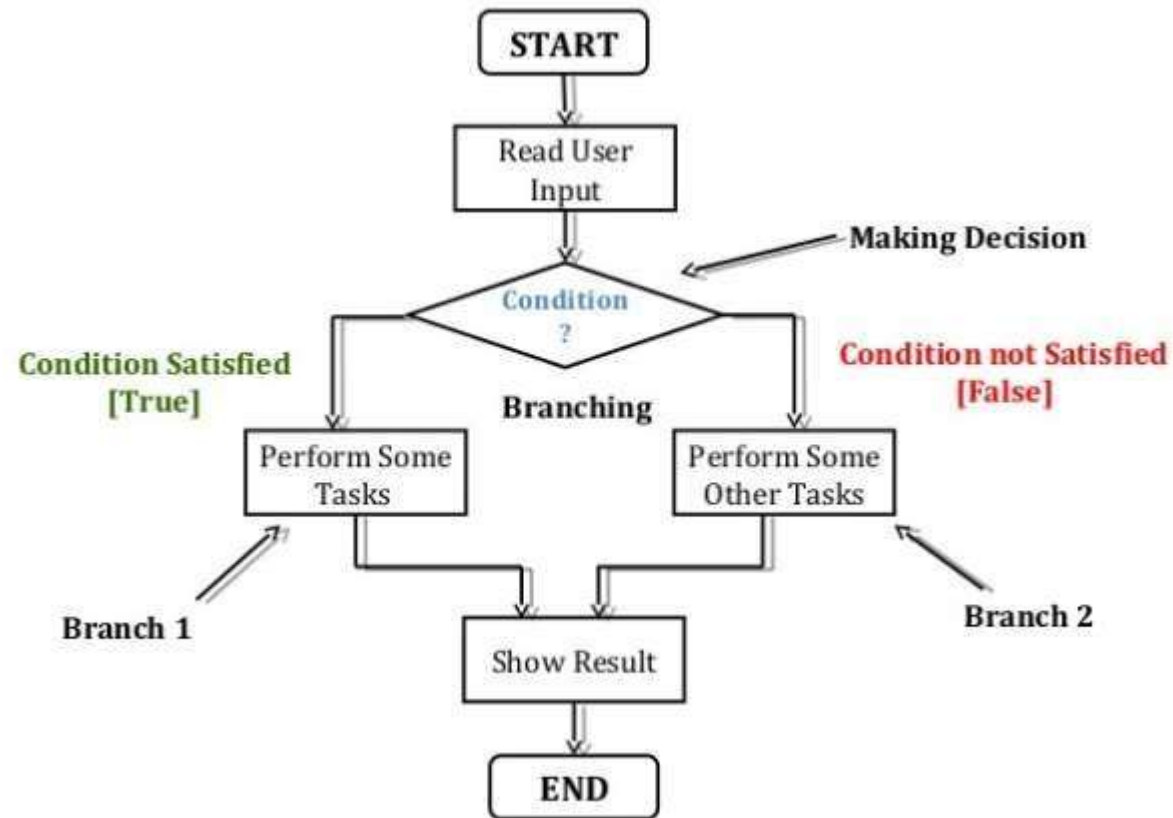
These statements ‘control’ the flow of execution they are also known as control statements.



# DECISION MAKING AND BRANCHING

## Decision making statement:

Decision making statements are used to skip or execute a group of statements based on the results of some condition.



# DECISION MAKING AND BRANCHING

## Decision making with simple if statement:

The if statement is used to control the flow of execution of statements.

If statement execute or skip one statement or group of statements for a particular condition.

General form:

```
if(text condition)
```

```
{
```

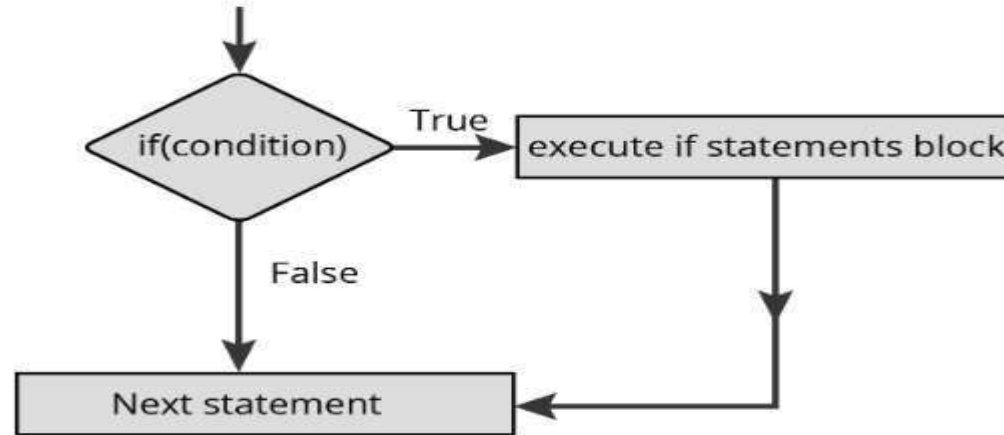
```
statement block;
```

```
}
```

```
next statement;
```

# DECISION MAKING AND BRANCHING

Decision making with simple if statement:



- When this statement is executed, the computer first evaluates the value of the test condition.
- If the value is true statement block and next statement are executed sequentially.
- If the value is false, statement block is skipped and execution starts from the next statement.

# DECISION MAKING AND BRANCHING

Decision making with simple if statement:

Rules:

- ✓ The brackets around the test condition are must.
- ✓ Test condition must be relational or logical expression.
- ✓ Statement block is called body of the if statement and it contains one or more statements.
- ✓ The opening and closed brackets { } are must if the statement block contains more than one statement. Else optional.

## DECISION MAKING WITH SIMPLE IF STATEMENT:

Example Program:

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

```
Void main()
```

```
{
```

```
int mark;
```

```
char grade;
```

```
scanf(“%d %c”, &mark, &grade);
```

```
if(grade==‘A)
```

```
{
```

```
mark=mark+10;
```

```
}
```

```
printf(“%d”, mark);
```

```
}
```