LECTURE 1

COMPONENTS OF A COMPUTER

Computer: Definition

A computer is a machine that can be programmed to manipulate symbols.

Its principal characteristics are:

- ✓It responds to a specific set of instructions in a well-defined manner.
- ✓It can execute a prerecorded list of instructions (a program).
- ✓It can quickly store and retrieve large amounts of data.
- ✓Therefore computers can perform complex and repetitive procedures quickly, precisely and reliably. Modern computers are electronic and digital. The actual machinery (wires, transistors, and circuits) is called hardware; the instructions and data are called software.

Computer sizes and power

Computers can be generally classified by size and power as follows, though there is considerable overlap:

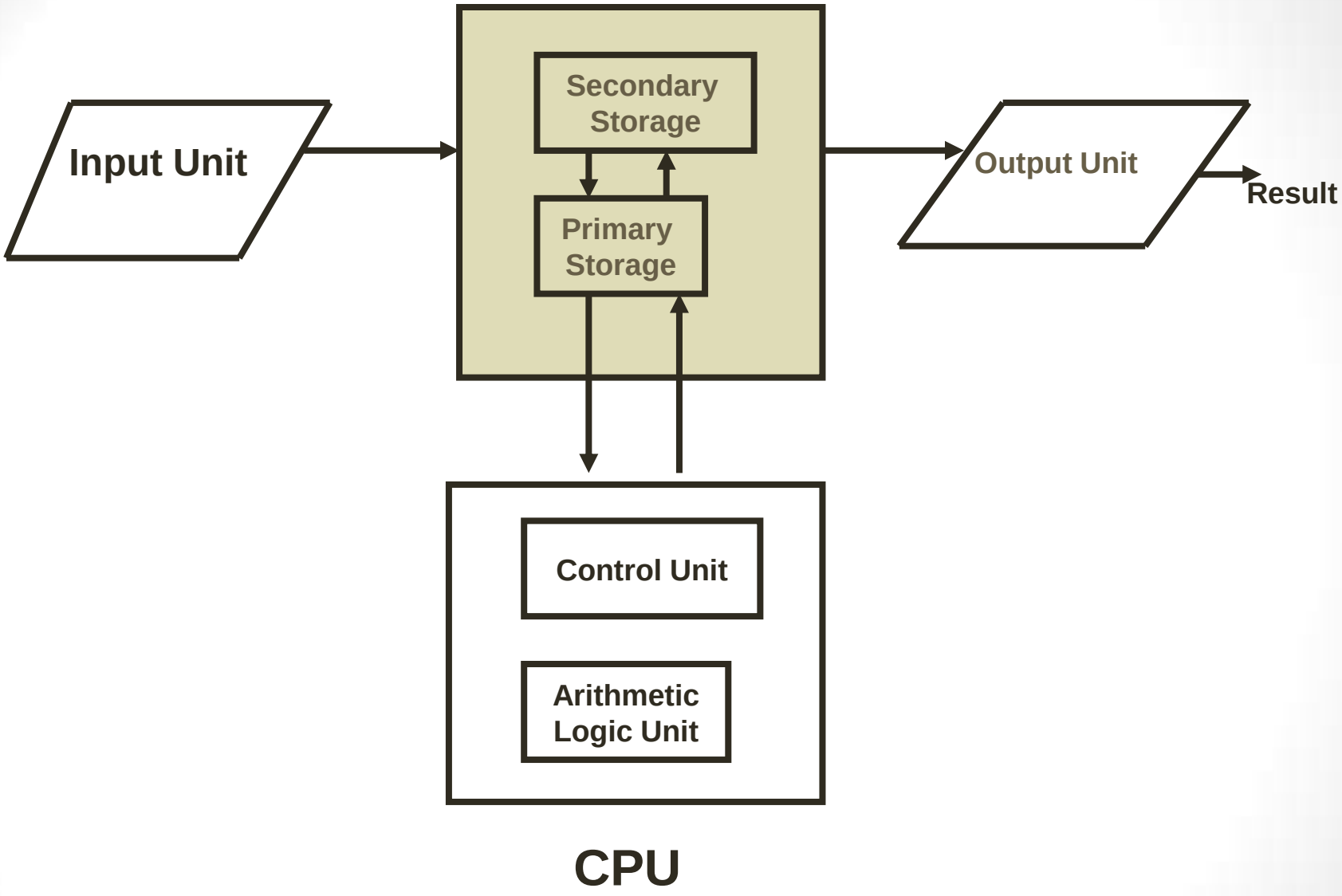
Personal computer: A small, single-user computer based on a microprocessor.

Workstation: A powerful, single-user computer. A workstation is like a personal computer, but it has a more powerful microprocessor and, in general, a higher-quality monitor.

Minicomputer: A multi-user computer capable of supporting up to hundreds of users simultaneously.

Mainframe: A powerful multi-user computer capable of supporting many hundreds or thousands of users simultaneously.

Supercomputer: An extremely fast computer that can perform hundreds of millions of instructions per second.



All general-purpose computers require the following hardware components:

Central processing unit (CPU): The heart of the computer, this is the component that actually executes instructions organized in programs ("software") which tell the computer what to do.

Memory (fast, expensive, short-term memory): Enables a computer to store, at least temporarily, data, programs, and intermediate results.

Mass storage device (slower, cheaper, long-term memory): Allows a computer to permanently retain large amounts of data and programs between jobs. Common mass storage devices include disk drives and tape drives.

Input device: Usually a keyboard and mouse, the input device is the conduit through which data and instructions enter a computer.

Output device: A display screen, printer, or other device that lets you see what the computer has accomplished.

In addition to these components, many others make it possible for the basic components to work together efficiently. For example, every computer requires a bus that transmits data from one part of the computer to another.

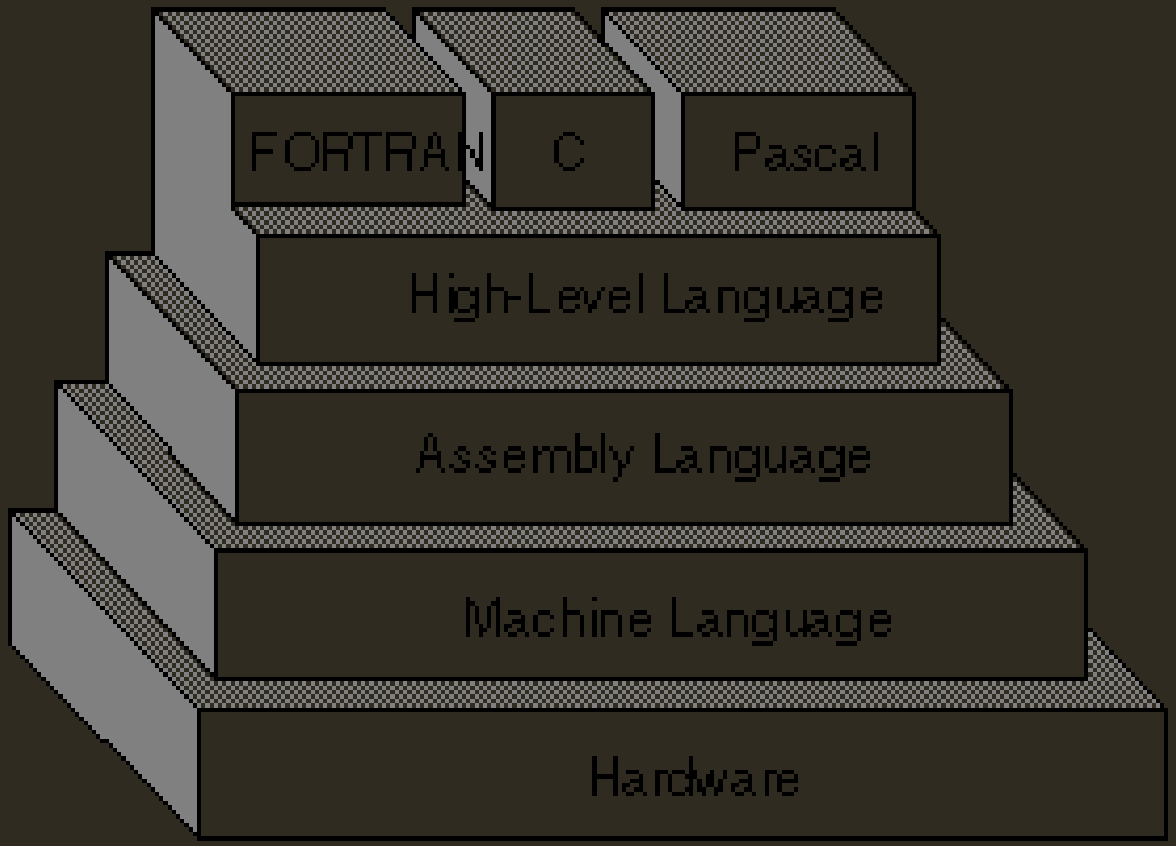
LECTURE 2

Computer Language

A system for communicating. Written languages use symbols (that is, characters) to build words. The entire set of words is the language's *vocabulary.* The ways in which the words can be meaningfully combined is defined by the language's *syntax* and *grammar.* The actual meaning of words and combinations of words is defined by the language's *semantics*.
In computer science, human languages are known as *natural languages*. Unfortunately, computers are not sophisticated enough to understand natural languages. As a result, we must communicate with computers using special computer languages.

Computer languages can be classified into :

1. Low level languages
2. High level languages



Machine Language

The lowest-level programming language (except for computers that utilize programmable microcode) Machine languages are the only languages understood by computers. While easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers. Programmers, therefore, use either a high-level programming language or an assembly language. An assembly language contains the same instructions as a machine language, but the instructions and variables have names instead of being just numbers.

Programs written in high-level languages are translated into assembly language or machine language by a compiler. Assembly language programs are translated into machine language by a program called an assembler.

Every CPU has its own unique machine language. Programs must be rewritten or recompiled, therefore, to run on different types of computers.

Assembly Language

A programming language that is once removed from a computer's machine language. Machine languages consist entirely of numbers and are almost impossible for humans to read and write. Assembly languages have the same structure and set of commands as machine languages, but they enable a programmer to use names instead of numbers.
Each type of CPU has its own machine language and assembly language, so an assembly language program written for one type of CPU won't run on another. In the early days of programming, all programs were written in assembly language. Now, most programs are written in a high-level language such as FORTRAN or C.
Programmers still use assembly language when speed is essential or when they need to perform an operation that isn't possible in a high-level language.

High Level Language

A programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from machine languages. In contrast, assembly languages are considered low-level because they are very close to machine languages.

The main advantage of high-level languages over low-level languages is that they are easier to read, write, and maintain. Ultimately, programs written in a high-level language must be translated into machine language by a compiler or interpreter.

The first high-level programming languages were designed in the 1950s. Now there are dozens of different languages, including Ada, Algol, BASIC, COBOL, C, C++, FORTRAN, LISP, Pascal, and Prolog.

LECTURE 3

FLOW CHART

Diagrammatic representation of the solution of a problem

Input/Output : Represented as a parallelogram. Examples: Get X from the user; usually containing the word "Start" or "End", or another phrase signaling the start or end of a process, such as "submit enquiry" or "receive product".

Arrows : Showing what's called "flow of display X.

Conditional (or **decision**) **:** Represented as a diamond (rhombus). These typically contain a Yes/No question or True/False test. This symbol is unique in that it has two arrows coming out of it, usually from the bottom point and right point, one corresponding to Yes or True, and one corresponding to No or False. The arrows should always be labeled. More than two arrows can be used, but this is normally a clear indicator that a complex decision is being taken, in which case it may need to be broken-down further, or replaced with the "pre-defined process" symbol.

A number of other symbols that have less universal currency, such as:

- A **Document** represented as a rectangle with a wavy base;

- A **Manual input** represented by rectangle, with the top irregularly sloping up from left to right. An example would be to signify data-entry from a form.

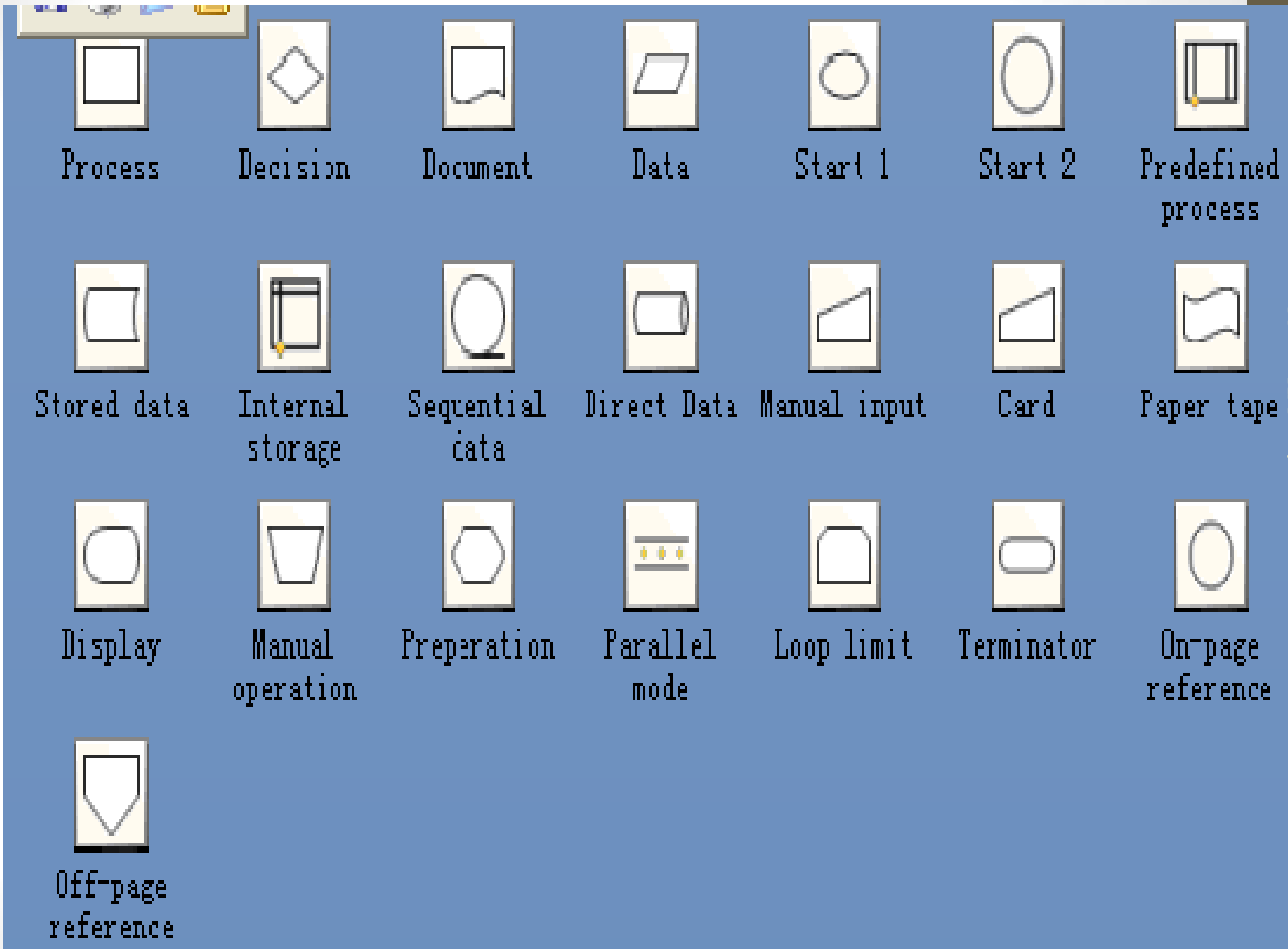
- A **Manual operation** represented by a trapezoid with the longest parallel side utmost, to represent an operation or adjustment to process that can only be made manually.

- A **Data File** represented by a cylinder

Note: All process symbols within a flowchart should be numbered. Normally a number is inserted inside the top of the shape to indicate which step the process is within the flowchart.

Flowcharts may contain other symbols, such as connectors, usually represented as circles, to represent converging paths in the flow chart. Circles will have more than one arrow coming into them but only one going out. Some flow charts may just have an arrow point to another arrow instead. These are useful to represent an iterative process (what in Computer Science is called a loop). A loop may, for example, consist of a connector where control first enters, processing steps, a conditional with one arrow exiting the loop, and one going back to the connector. Off-page connectors are often used to signify a connection to a (part of a) process held on another sheet or screen. It is important to remember to keep these connections logical in order. All processes should flow from top to bottom and left to right.

A flowchart is described as "cross-functional" when the page is divided into different "lanes" describing the control of different organizational units. A symbol appearing in a particular "lane" is within the control of that organizational unit. This technique allows the analyst to locate the responsibility for performing an action or making a decision correctly, allowing the relationship between different organizational units with responsibility over a single process.



LECTURE 4

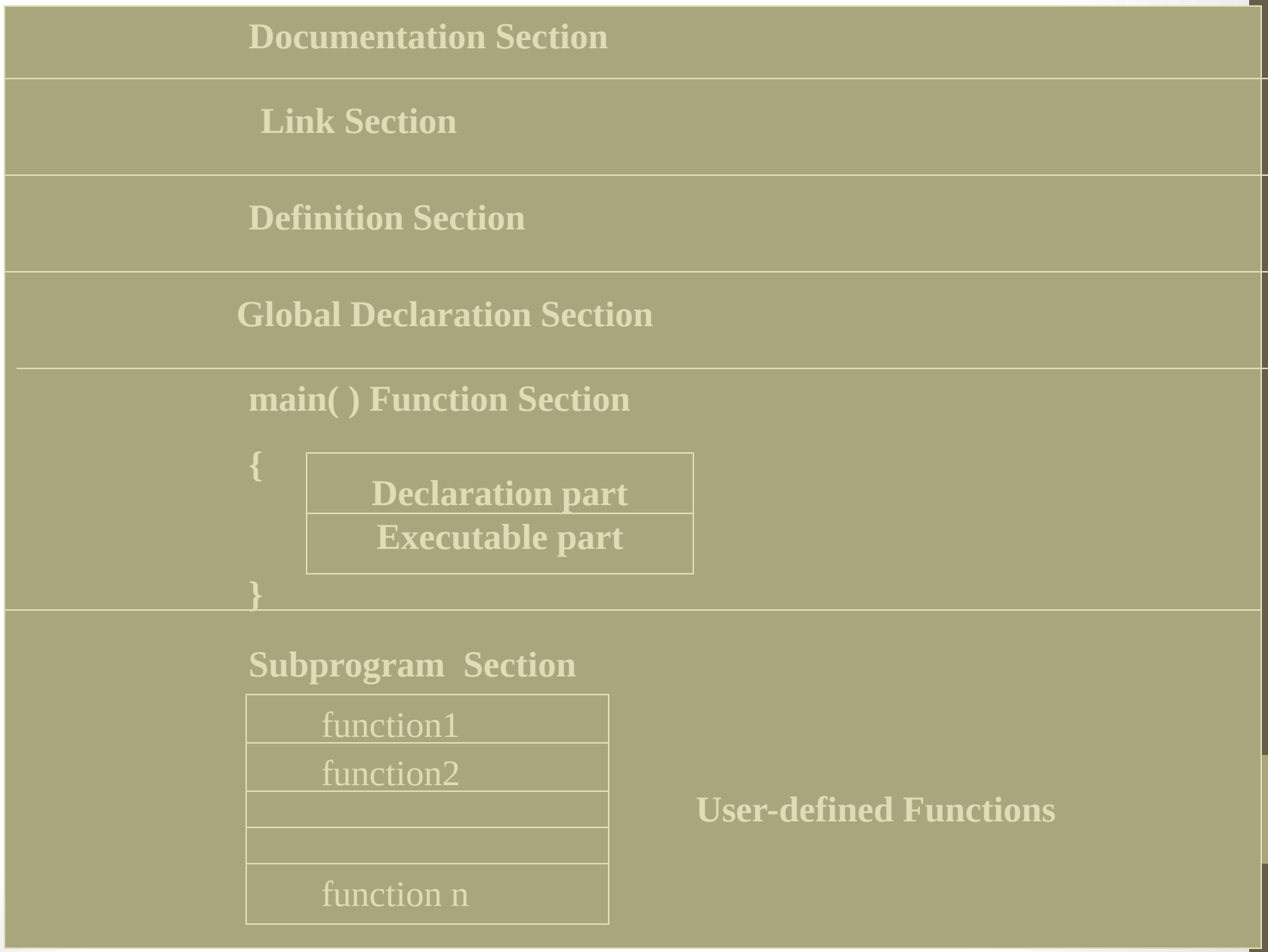
INTRODUCTION TO C

HISTORY OF C

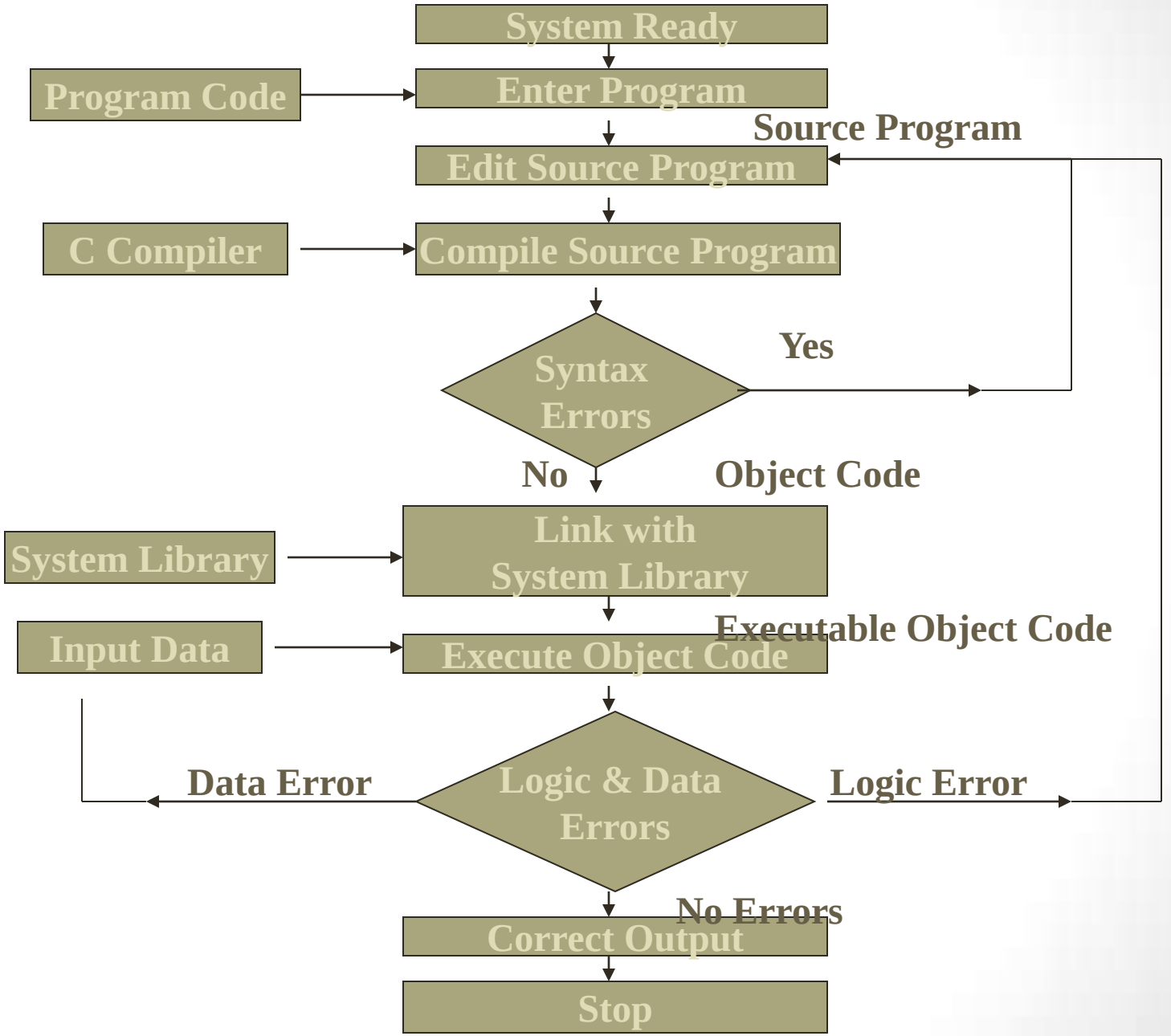
IMPORTANCE OF C

- **Programs written in C are efficient & fast.**
- **Its strength lies in built-in functions which can be used for developing programs.**
- **C is highly portable it means C programs written for one computer can be run on another with little or no modification.**
- **It is well suited for structured programming. User think of a problem in terms of function modules or blocks. A proper collection of these modules would make a complete program. It makes program debugging, testing and maintenance easier.**
- **C has the ability to extend itself.We can add our own functions to C library.**
- **C compiler combines the capabilities of an assembly language with the features of high-level languages therefore it is well suited for writing both system software and business packages.**

BASIC STRUCTURE OF C PROGRAM



EXECUTING A C PROGRAM



CHARACTER SET

In C the characters that can be used to form words, numbers and expressions are grouped into the following categories.

1. LETTERS : **Uppercase A……Z** **Lowercase a……z**

2. DIGITS : **All decimal digits 0……9**

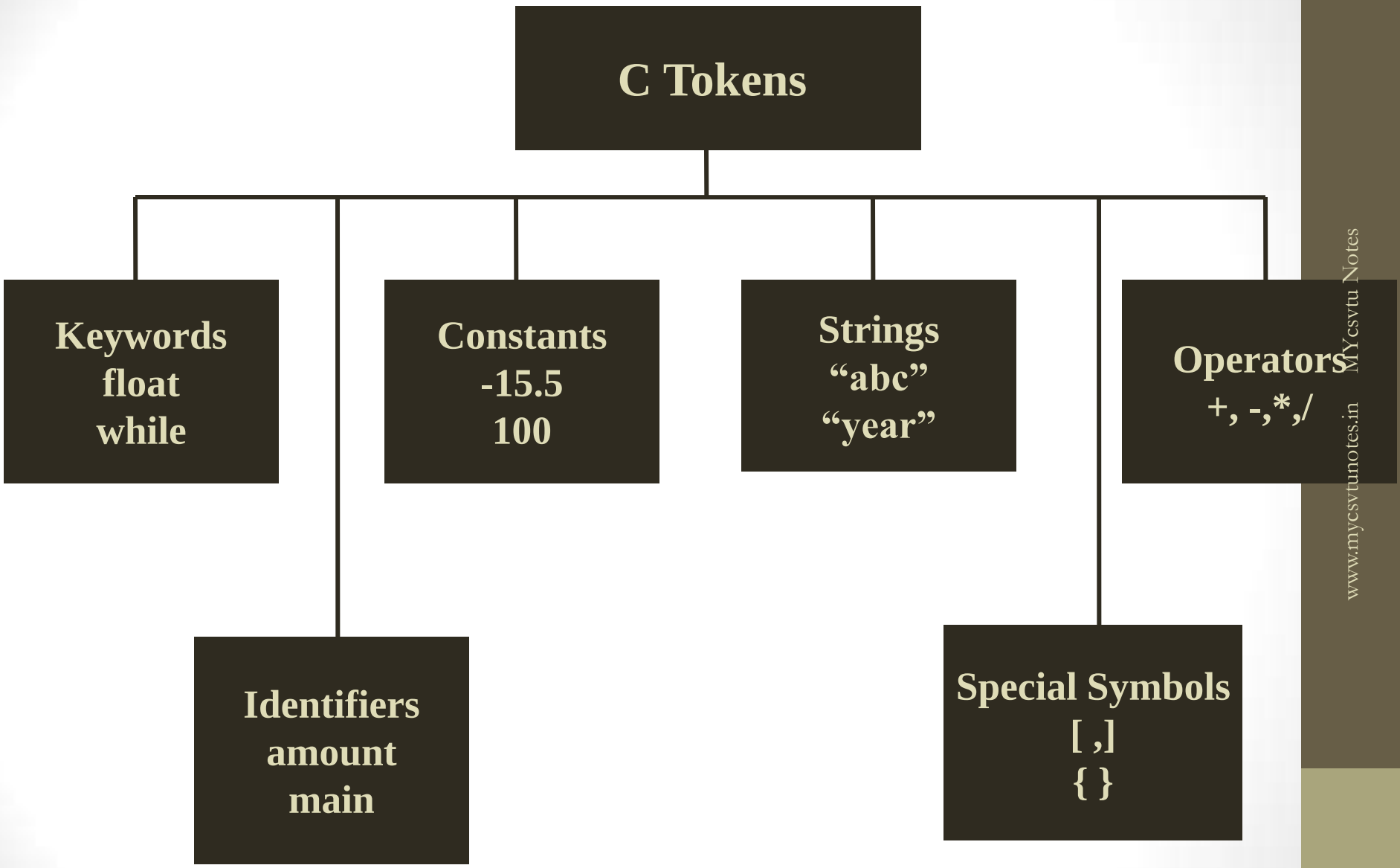
3. SPECIAL CHARACTERS :

, comma	\ backslash	~ tilde
& ampersand	_ under score	$ dollar sign
. Period	% percent sign	^ caret
; semicolon	* asterisk	- minus sign
: colon	+ plus sign	< opening angle bracket
? Question mark	(left parenthesis	> closing angle bracket
‘ apostrophe	) right parenthesis	[left bracket
“ quotation mark	{ left brace	] right bracket
! exclamation	} right brace	# number sign
| vertical bar	/ slash	

4. WHITE SPACES : Blank space, Horizontal tab, Carriage sign, New line , Form feed

C TOKENS

In a C program the smallest individual units are known as C tokens. C programs are written using these tokens and the syntax of the language.



Keywords :

keywords are the words whose meaning has already been defined by the C compiler. All keywords have fixed meanings and these meanings cannot be changed. Keywords serve as basic building blocks for program statements. All keywords must be written in lowercase.

For Example :

char, const, int, float, for, struct, union, switch, goto, while

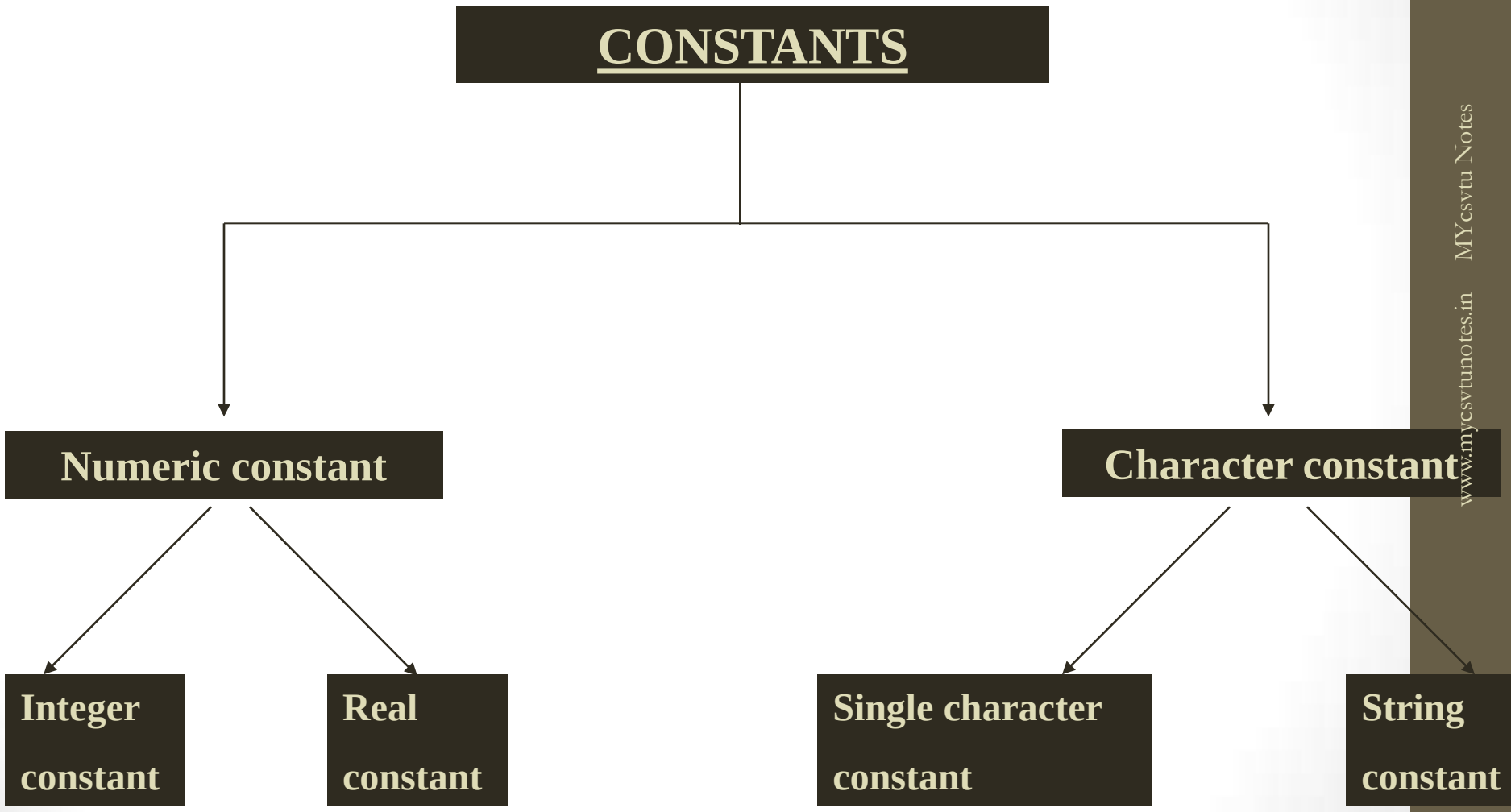
Identifiers : identifiers refer to the names of the variables, functions and arrays. These are the user-defined names and consist of sequence of letters and digits, with a letter as a first character.

For Example : Sum, Total, count.

Rules for identifiers :

1. First character must be an alphabet or underscore.
2. Must consist of only letters, digits or underscore.
3. Only first 31 characters are significant.
4. Cannot use a keyword.
5. Must not contain white space.

Fixed values that do not change during the execution of a program.



Numeric constants

Integer constants An integer constant refers to a sequence of digits.

There are three types of integer constants :

Decimal integer : It consist of a set of digits 0 through 9, preceded by an optional – or + sign

123, -123, 0 65478

Embedded spaces commas and non digit characters are not permitted

Example : 20,000, 15 750 , $1000

Octal integer : Combination 0f digits 0 through 7 with a leading 0.

For example: 037, 0, 0435,0551

Hexadecimal integer : Sequence of digits preceded by 0x or 0X they may include alphabets A through F or a through f.

For example : 0X2, 0x9F,0Xbcd

Real constants

Numbers containing fractional parts are called real (floating point) constants.

For example : 0.0083, -0.75, 435.36

Character constants

Single character constant

Single character enclosed within a pair of single quotes.

For example : ‘5’, ‘X’

String constant

Sequence of characters enclosed within a pair of double quotes.

For example : “hello”, “1234”, “well done”, “X”

Backslash Character constants

C supports some special backslash character constants that are used in output functions.

‘\a’	audible alert
‘\b’	blank space
‘\f’	form feed
‘\n’	new line
‘\r’	carriage return
‘\t’	horizontal tab
‘\v’	vertical tab
‘\’’	single quote
‘\”’	double quote
‘\?’	question mark
‘\\’	backslash
‘\0’	null

LECTURE 5

DATA

TYPES

Data types in C

C language is rich in data types. The variety of data types available allow the programmer to select the type appropriate to the needs of the application as well as machine.

C supports three classes of data types :

1 Primary (or fundamental) data types

2. Derived data types

3. User-defined data types

PRIMARY DATA TYPES

Integer Type :

Integer	
signed	unsigned
int	unsigned int
short	unsigned short int
long int	unsigned long int

Character
char
signed char
unsigned char

Floating point type		
float	double	long double

void

Type	Size	Range
char or signed char	8	-128 to 127
unsigned char	8	0 to 255
int or signed int	16	-32,768 to 32,767
unsigned int	16	0 to 65535
short int or signed short int	8	-128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	-2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
long double	80	3.4E-4932 to 1.1E+4932

void types : The void has no values. This is usually used to specify the type of functions. The type of function is said to be void when it does not return any value to calling function. It can also play the role of a generic type, meaning that it can represent any of the other standard types.

Derived data types : arrays, functions, structures and pointers

USER DEFINED TYPE DECLARATION

C supports a feature known as “type definition” that allows users to define an identifier that would represent an exiting data type. The user-defined data type identifier can later be used to declare variables.

It takes the general form :

typedef type identifier ;

Where type refers to an exiting data type and identifier refers to new name given to the data type. The exiting data type may belong to any class of type , including the user-defined ones.

The main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

Another user-defined data type is “enumerated data type” provided by ANSI standard.

It is defined as follows:

enum identifier {value1,value2,……,valuen};

The “identifier” is a user-defined enumerated data type which can be used to declare variables that can have one of the values enclosed with in the braces. Known as enumeration constants .

Syntax used to declare variables to be of this new type :

enum identifier v1,v2,……,vn;

The definition and declaration of enumerated variables can be combined in one statement.

e.g. enum day {Monday, Tuesday,…… ,Sunday} week_st, week_end;

TYPE CONVERSION IN EXPRESSIONS

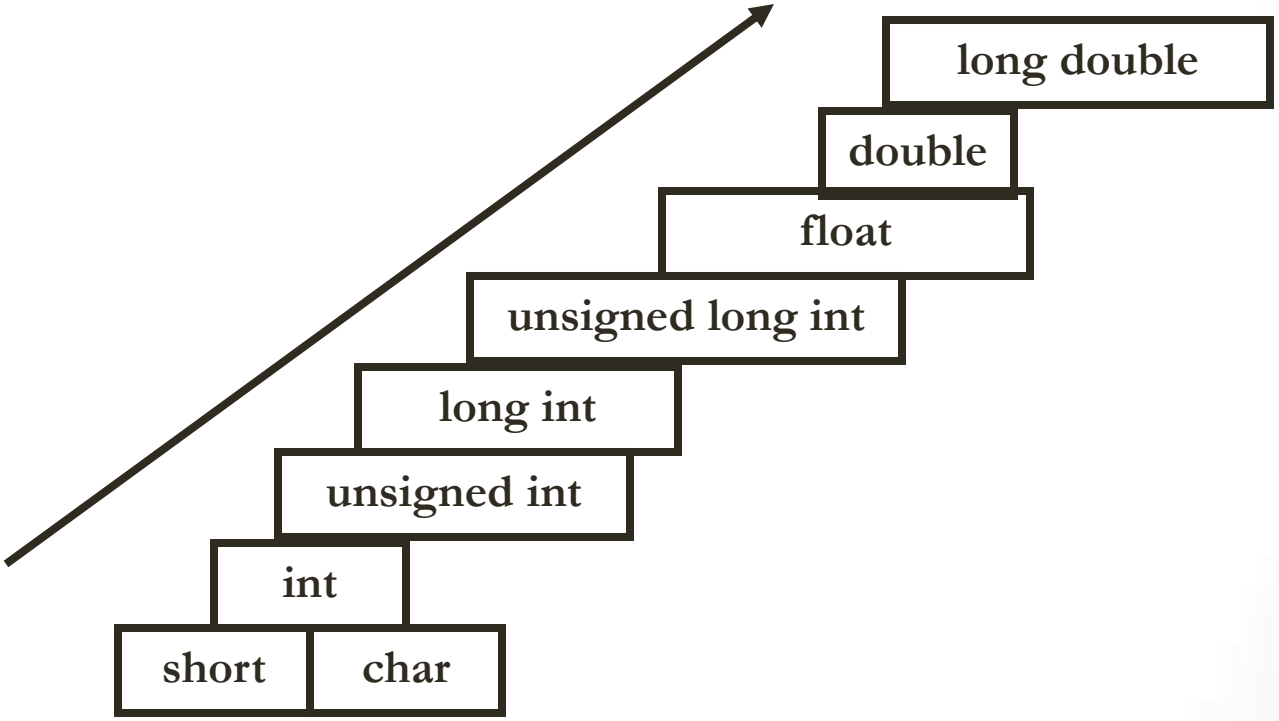
C permits mixing of constants and variables of different type in an expression.

C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance.

This automatic conversion is known as implicit type conversion.

C uses the rule that, in all expressions except assignments, any implicit type conversions are made from a lower size type to a higher size type.

CONVERSION HIERARCHY



EXPLICIT CONVERSION

C performs type conversion automatically. However there are instances when we want to force a type conversion in a way that is different from the automatic conversion.

This type of local conversion is known as explicit conversion or casting a value.

The general form of a cast is :

(type-name)expression

For example :

b= double(sum)/n

a=(int)21.3/(int)4.5

y=(int)(a+b)

VARIABLES

- A variable is a data name that may be used to store a data value.
- A variable may take different values at different times during execution.
- A variable name may be chosen in a meaningful way so as to reflect its function or nature in program.
- A variable name may consist of letters, digits, and underscore _ character.

For Example : average, sum, Total, count, breadth, length, sum_even, class_strength

Rules for forming a variable

- ✓ They must begin with a letter.
- ✓ Length should not be normally more than eight characters.
- ✓ Uppercase and lowercase are significant. That is Total is not same as total or TOTAL.
- ✓ It should not be a keyword.
- ✓ White space is not allowed.

Declaring a variable

After assigning a suitable variable names, we must declare them to the compiler.

Declaration does two things :

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

The syntax for declaring a variable is as follows :

```
data-type v1,v2,……,vn;
```

Where v1,v2,……,vn are the names of variables. Variables are separated by commas. A declaration statement must end with a semicolon.

data-type v1,v2,……,vn;

For example :

```
int count;

float number,total;

double ratio;

int i=0,j=0;
```

LECTURE 6

OPERATORS

OPERATORS

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulation. Operators are used in programs to manipulate data and variables. They usually form a part of the mathematical or logical expression.

C operators can be classified into following categories

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and decrement operators
6. Conditional operators
7. Bitwise operators
8. Special operators

ARITHMETIC OPERATORS

+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Example : a+b, a-b, a*b, a/b, a%b

Here a & b are known as operands

RELATIONAL OPERATORS

These are the operators used to compare quantities.

For example: to compare age of two persons or the price of two quantities.

<	**is less than**
<=	**is less than or equal to**
>	**is greater than**
>=	**is greater than or equal to**
==	**is equal to**
!=	**is not equal to**

For example : age1>=age2, sal1<=sal2, x!=y

ASSIGNMENT OPERATOR =

Assignment operators are used to assign the result of an expression to a variable.
For example : c=a+b

In addition to to usual '=' operator C has a set of shorthand assignment operator of the form

v op = exp;

Where v is a variable, exp is an expression and op is a C binary arithmetic operator.

It is equivalent to v=v op (exp);

op= is known as the shorthand assignment operator.

For example : a*=b means a=a*b

a+=b means a=a+b

a/=b means a=a/b

LOGICAL OPERATORS

C has three logical operators

&&	**AND**
||	**OR**
!	**NOT**

The Logical operator && and || are used when we want to test more than one condition and make decisions

```
if(age>55 && salary<1000)

if(number<0||number>100)
```

INCREMENT & DECREMENT OPERATOR

++ adds 1 to operand

-- subtracts 1

Both are unary operator and takes the following form:

++m or m++ equivalent to m=m+1
--m or m-- equivalent to m=m-1

Example : m=5 than y=++m;

In this case the value of *y* and *m* would be *6*

y=m++;

In this case value of y would be 5 and m would be 6

RULES FOR ++ AND -- OPERATOR

- **Increment and decrement operators are unary operators and they require variable as their operands.**
- **When postfix ++ or - - is used with a variable in an expression , the expression is evaluated first using the original value of the variable and then the variable is incremented or decremented by one.**
- **When prefix ++ or - - is used in an expression , the variable is incremented or decremented first and then the expression is evaluated using the new value of the variable.**
- **The precedence and associatively of ++ and - - operators are the same as those of unary + and unary -**

CONDITIONAL OPERATOR

A ternary operator pair "?:" is available in C to construct conditional expression of the form :

exp1 ? exp2 : exp3

Where exp1,exp2,exp3 are expressions.

For eg:

```
a=10;
b=15;

x=(a>b) ?  a : b ;

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

BITWISE OPERATORS

These are the operators that manipulate data at bit level.

&	**bitwise AND**
|	**bitwise OR**
^	**bitwise exclusive OR**
<<	**shift left**
>>	**shift right**

Example : a=4=0100

b=3=0011

a | b =0111

X=0100 1001 1100 1011

X<<3 = 0100 1110 0101 1000

X>>3 = 0000 1001 0011 1001

~X = 1011 0110 0011 0100

OPERATOR PRECEDENCE AND ASSOCIATIVITY

Each operator in C has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct level of precedence and an operator may belong to one of these levels. The operator at high level of precedence are evaluated first. The operators of the same precedence are evaluated from left to right or from right to left depending on the level.

This is known as associativity property of an operator.

SPECIAL OPERATORS

sizeof Operator – Returns number of bytes the operand occupies.

Example : m=sizeof(sum);

Comma Operator - Used to link the related expression together.

Example : value=(x=10,y=5,x+y);

Pointer Operators – & address of

*** value at**

Member Selection Operators - . and –>

RULES OF PRECEDENCE AND ASSOCIATIVITY

- Precedence rules decodes the order in which different operators are applied.
- Associativity rule decodes the order in which multiple occurrence of the same level operator are applied.

Arithmetic	*****	**Multiplication**
	/	**Division**
	%	**Reminder (modulus)**
	+	**Binary plus**
	-	**Binary minus**

Associativity : L-> R

For example :

X=a-b/3+c*2-1 **a=9, b=12, c=3**

9-12/3+3*2-1

9-4+3*2-1

9-4+6-1

5+6-1

11-1

10

Expression

An expression is a combination of variables, constants and operators written according to the syntax of the language.

In C every expression evaluates to a value i.e. every expression results in some value of a certain type that can be assigned to a variable.

Variable=expression

For example : x=a*b-c

z=a-b/c+d

Rules for evaluation of Expression

1. First, parenthesized sub expression from left to right are evaluated.
2. If parentheses are nested, the evaluation begins with the innermost sub-expression.
3. The precedence rule is applied in determining the order of application of operators in evaluating sub-expressions
4. The associativity rule is applied when two or more operators of the same precedence level appear in a sub-expression.
5. Arithmetic expressions are evaluated from left to right using the rules of precedence.
6. When parentheses are used, the expressions within parentheses assume highest priority.

LECTURE 7

INPUT/OUTPUT OPERATIONS

Input/output operations in C :

Unlike other high-level languages, C does not have any built-in input/output statements as part of its syntax. All input/output operations are carried out through function calls such as printf and scanf. There exist several functions that have more or less become standard for input and output operations in C. Those functions are collectively known as the standard I/O library.
Each program that uses a 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. This instruction tells the compiler to search for a file named stdio.h and place its contents at this point in program. The contents of the header file become part of the source code when it is compiled.

Reading a character :

The simplest of all input/output operations is reading a character from the standard input unit (usually the keyboard) and writing it to the standard output unit (usually the screen).

Reading a single character can be done by using the function getchar.

variable_name=getchar( );

variable_name is a valid C name that has been declared as char type.

When this statement is encountered, the computer waits until a key is pressed and then assigns this character as a value to getchar function. Since getchar is used on the right hand side of an assignment statement, the character value of getchar in turn assigned to the variable on the left.

For example :

```
char choice;
choice=getchar( );
```

Writing a character :

Writing a single character to the terminal can be done by using the function puchar.

It takes the following form :

putchar(variable_name);

Where variable name is a type char variable containing a character.

This statement displays the character contained in the variable at the terminal.

For example :

char choice='y';

putchar(choice);

LECTURE 8

FORMATTED INPUT AND OUTPUT

Formatted Input

Formatted input refers to an input data that has been arranged in a particular format.

For example : 15.75 123 jhon

Here first part should be read into a variable float, the second into int, the third part into char.

This is possible in C using scanf function (scanf means scan formatted).

The general form of scanf is :

scanf("control string", arg1, arg2, arg3,....., argn);

The control string specifies the field format in which the data is to be entered and the arguments arg1, arg2, arg3,....., argn specify the address of locations where the data is stored. Control string and arguments are separated by commas.

Control string also known as format string contains field specifications, which direct the interpretation of input data. It may include :

1. **Field (or format) specifications, consisting of the conversion character %, a data type character (or type specifier), and an optional number specifying the field width.**
2. **Blanks, tabs or newlines.**

Blanks, tabs and newlines are ignored. The data type character indicates the type of data that is to assigned to the variable associated with the corresponding argument. The fiel width specifier is optional.

Commonly used scanf format codes

Code	Meaning
%c	reads a single character
%d	reads a decimal integer
%e	reads a floating point value
%f	
%g	
%h	
%i	
%o	
%s	
%u	
%x	
%[..]	

Scanf width specifier

The field specification for reading an integer is :

%wd

Here w is an integer number that specifies the field width of the number to be read.

For example : scanf("%2d%5d",&num1, &num2);

FORMATTED OUTPUT

It is desirable that the outputs are produced in such a way that they are understandable and are in an easy-to-use form.

The printf statement provides certain features that can be effectively exploited to control the alignment and spacing of print outs on the terminal.

The general form of printf statement is :

Printf(“control string”, arg1, arg2, arg3,……., argn);

Control string consists of three types of items :

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

The control string indicates how many arguments follow and what their types are. The arguments arg1, arg2, arg3,……., argn are the variables whose values are formatted and printed according to the specifications of the control string. The arguments should match in number, order and type with the format specifications.

Output of integer numbers

The format specification for printing an integer number is :

%wd

- **Where w specifies the minimum field width for the output.**

- **However, if a number is greater then the specification field width, it will be printed in full, overriding the minimum specification.**
- **d specifies that the value to be printed is an integr. The number is written in right-justified in the given field width. Leading blanks will appear as necessary.**

int x=9876;

printf(“%d”,x);

printf(“%6d”,x);

printf(“%2d”,x);

It is possible to force the printing to be left-justified by placing a minus sign directly after the % character.

printf(“%-6d”,x);

Output of real numbers

The format specification for printing a real number is :

%w.pf

- **The integer w indicates the minimum number of positions that are to be used for the display of the value**
- **The integer p indicates the number of digits to be displayed after the decimal point (precision).**
- **The value when displayed , is rounded to p decimal places and printed right-justified in the field of w columns.**

```
float y=98.7654;

printf("%7.4",y);

printf("%7.2f",y);

printf("%-7.2f",y);

printf("%f",y);

printf("%10.2e",y);

printf("%11.4e",-y);

printf("%-10.2e",y);

printf("%e",y);
```

Printing a single character

A single character can be displayed in a desired position using the format :

%wc

The character will be displayed right-justified in the field of w columns.

We can make the display left-justified by placing a minus sign before the integer w. the default value for w is 1.

Printing of strings

The format specification for outputting string is :

%wp.s

Where w specifies the field width for display and p instructs that only the first p characters of the string are to be displayed. The display is right-justified.

```
printf("%s",s1);

printf("%20s",s1);

printf("%20.10s",s1);

printf("%.5s",s1);

printf("%-20.10s",s1);

printf("%5s",s1);
```

LECTURE 9

DECISION MAKING AND BRANCHING

There may be situations where we want to change the order of execution of statements based on certain decisions or repeat a group of statement until specified conditions are met. This involves a kind of decision making to see whether a particular condition has occurred or not and then direct the computer to execute certain statements accordingly.

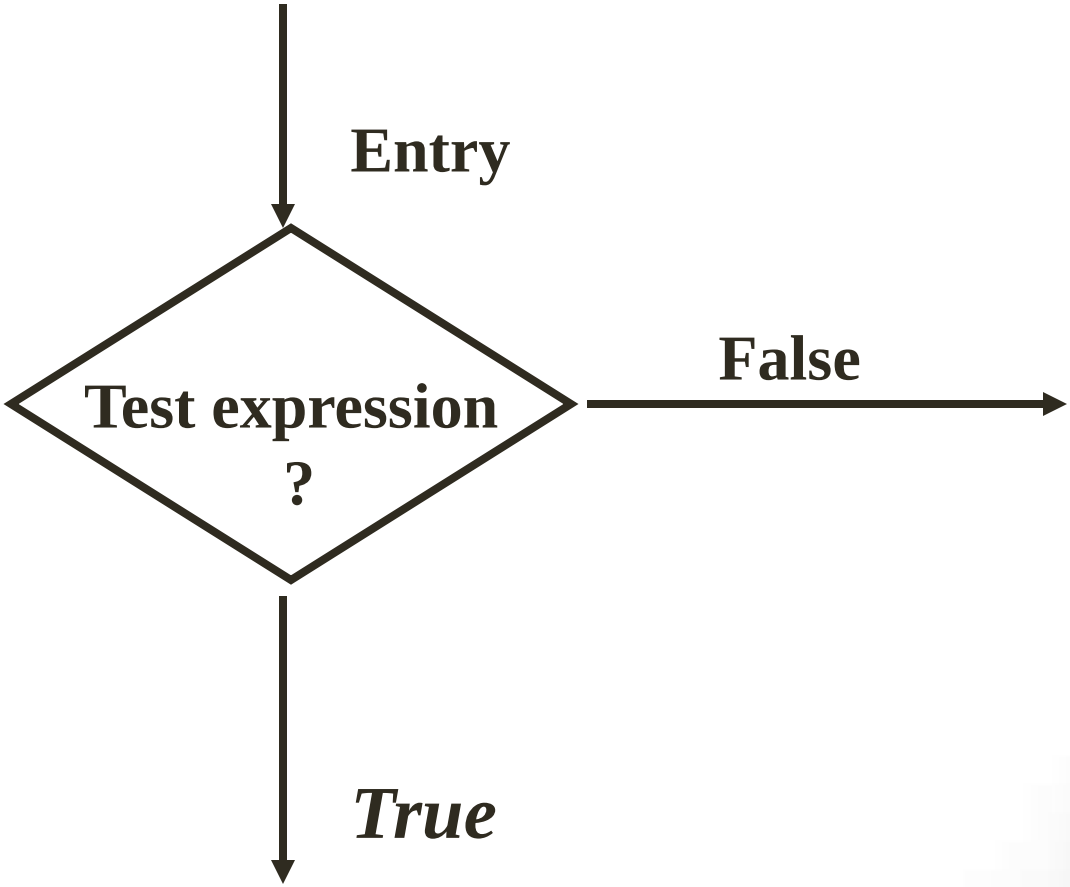
C possesses decision making by supporting following statements :

1. **if statement**
2. **switch statement**
3. **Conditional operator statement**
4. **goto statement**

These statements are popularly known as decision-making statement. Since these statements control the flow of execution, they are also known as control statements.

if statement

It is a basically two-way decision statement and is used in conjunction with an expression.



Different forms of if statement are :

1. Simple if statement

2. if……else statement

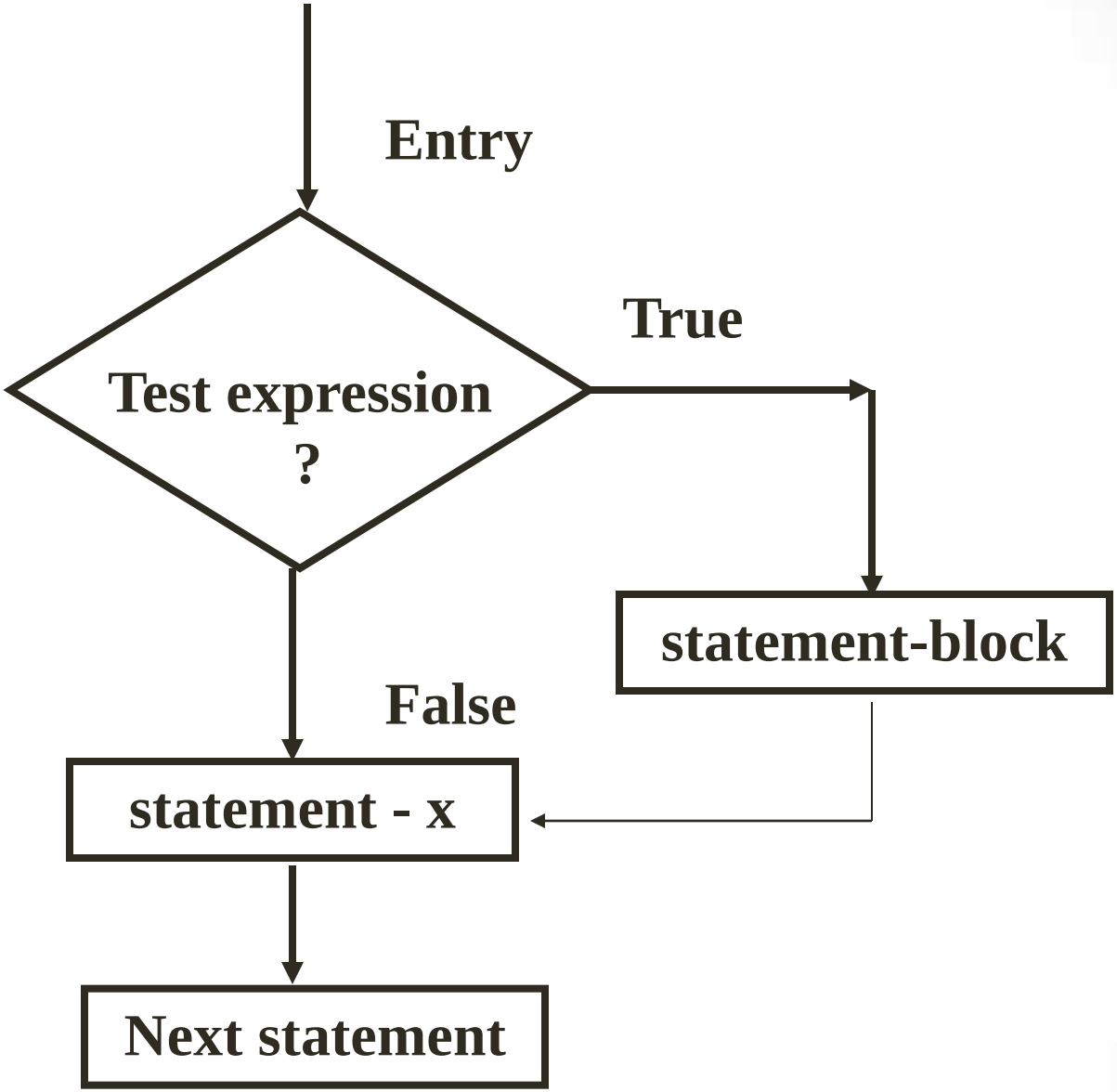
3. Nested if……..else statement

4. else if ladder

Simple if statement

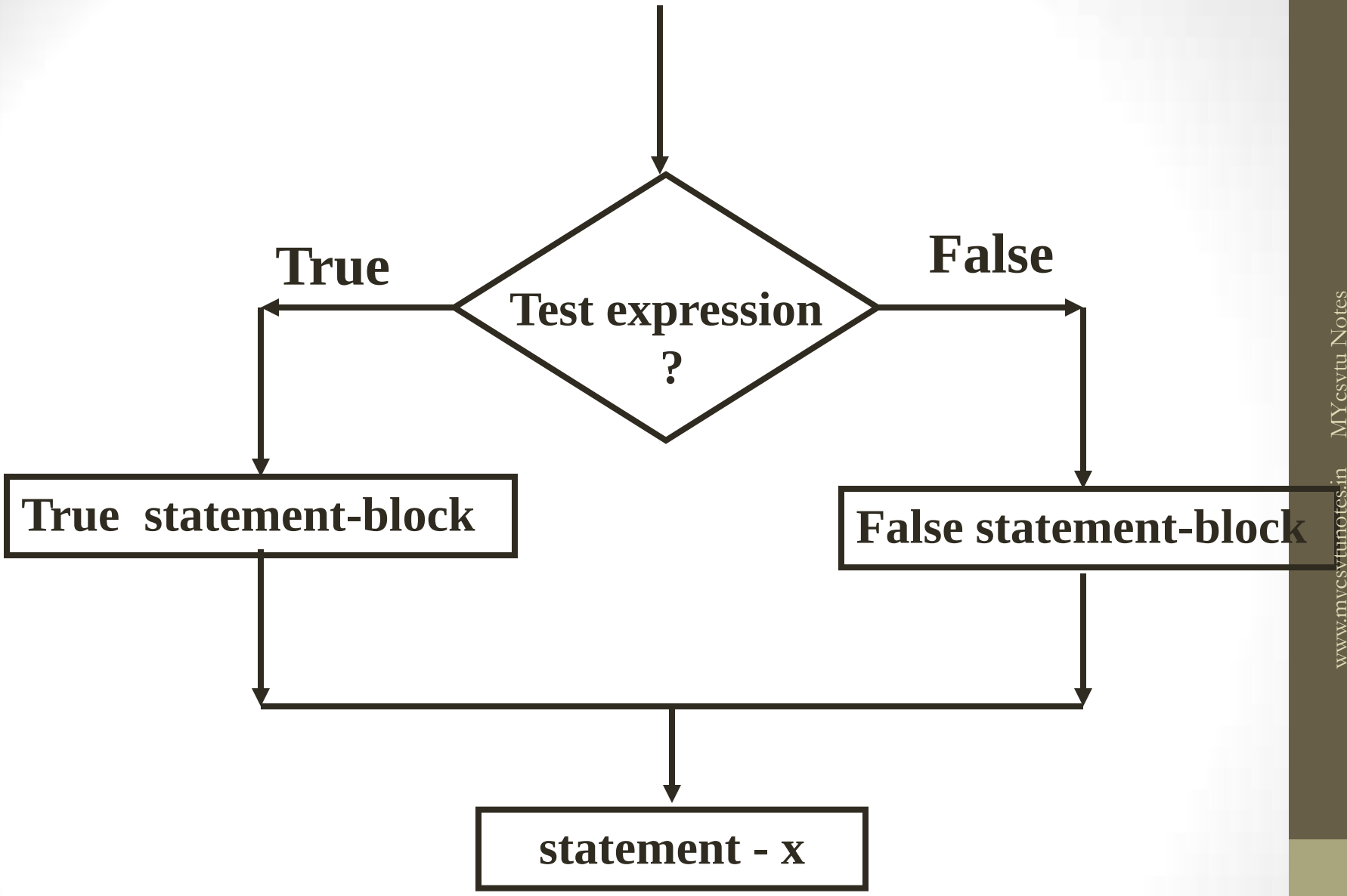
It takes the following form :

```
if(test expression)
{
   statement-block;
}
    statement x;
```



if……else statement

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



```
if(n>=0)
sq=sqrt(n);
else
printf("not possible");
```

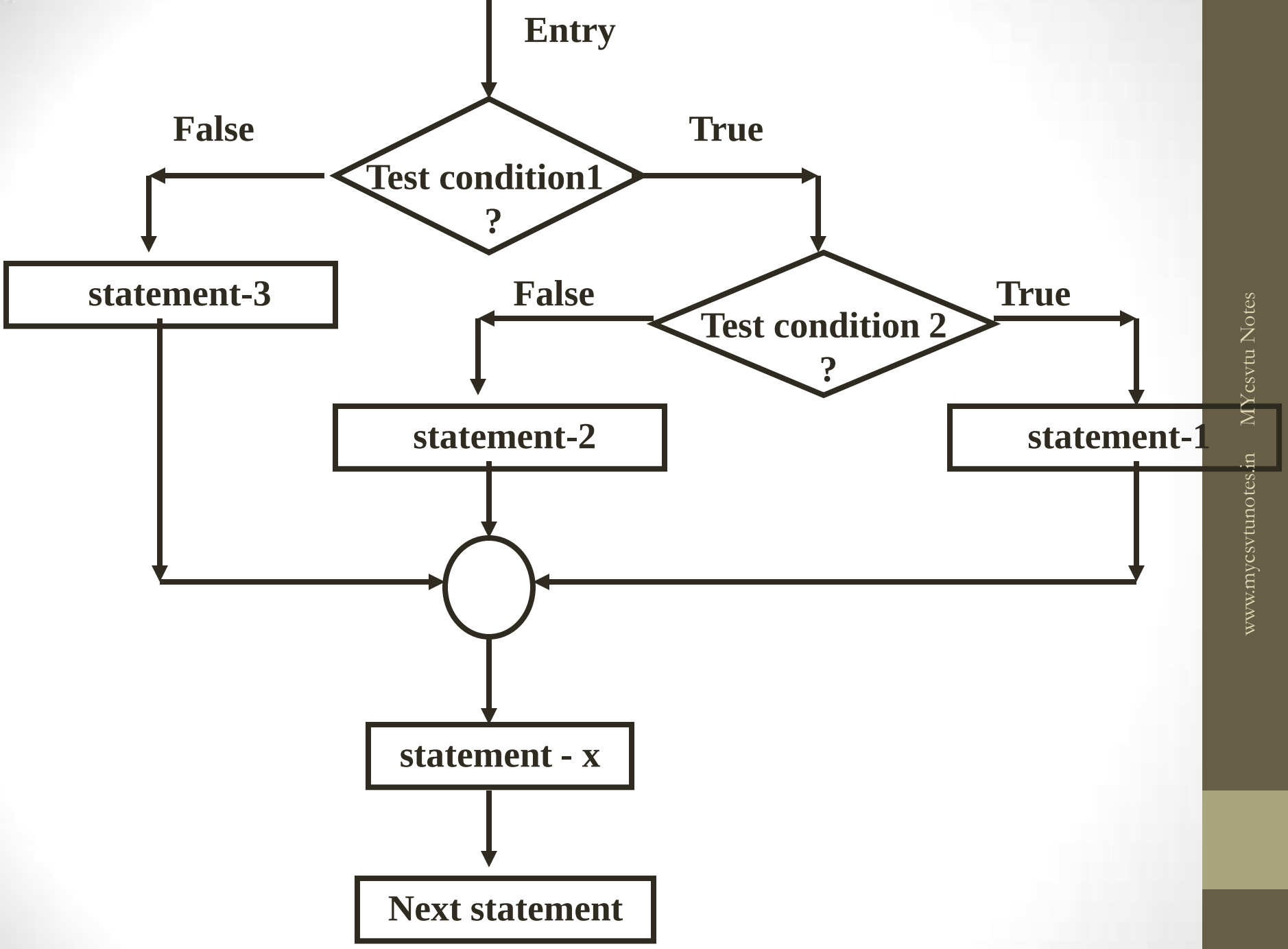
LECTURE 10

DECISION MAKING AND BRANCHING

Nested if…….else statement

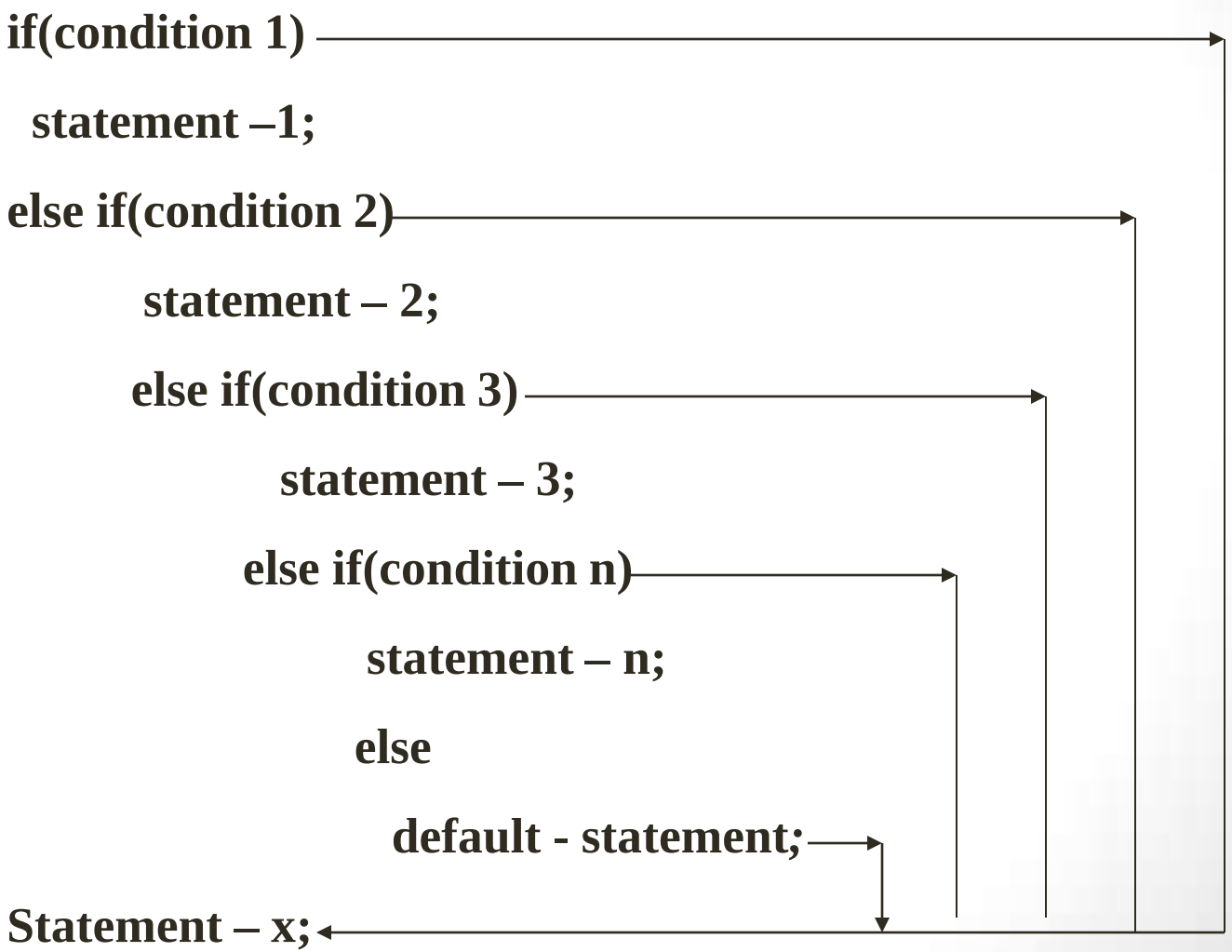
When a series of decisions are involved, we may have to use more than one if…….else statement in nested form.

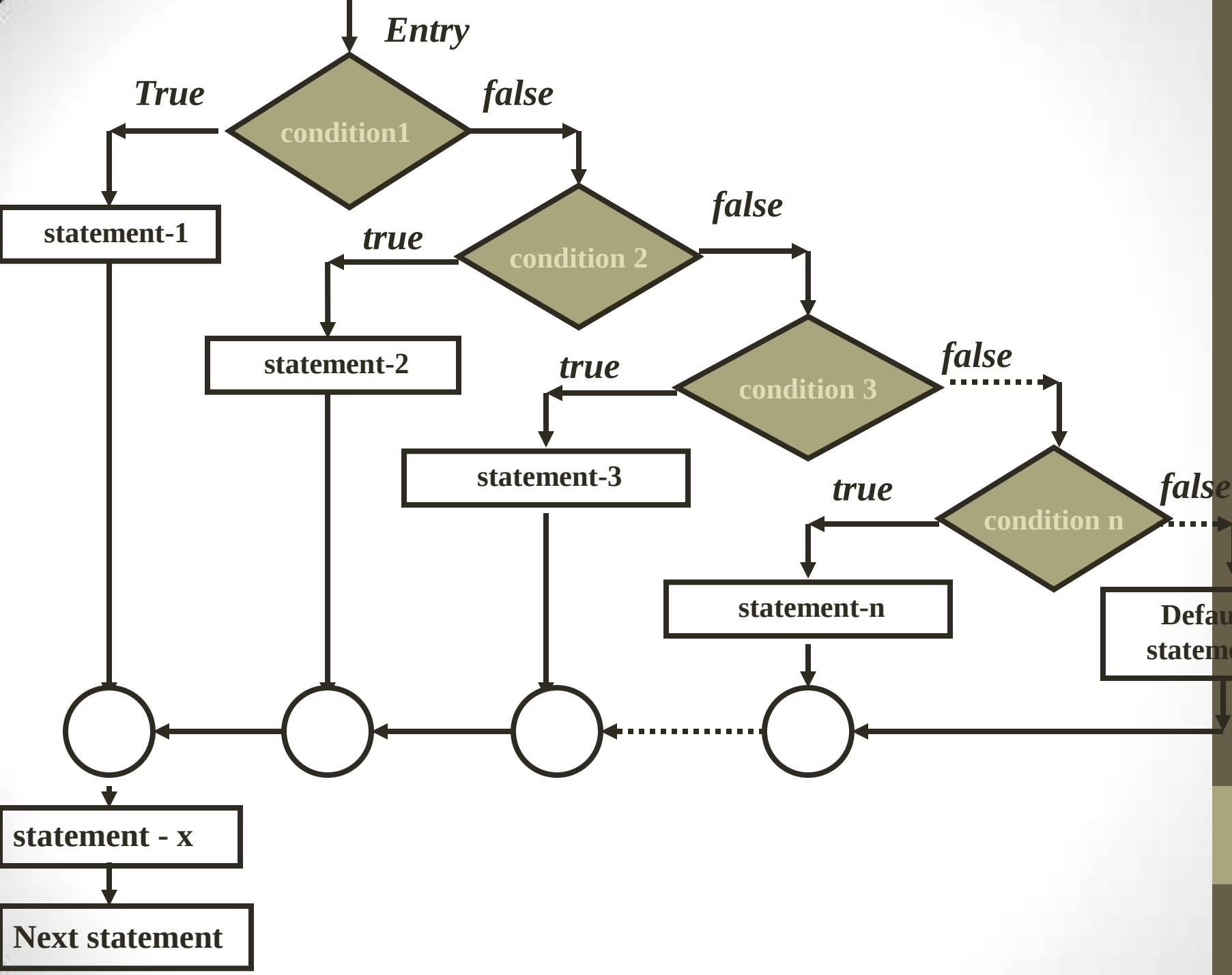
```
if(test condition-1)
{
    if(test condition-2)
    {
        statement-1;
     }
    else
    {
        statement-2;
     }
 else
{
   statement-3;
}
statement-x;
```



else if ladder

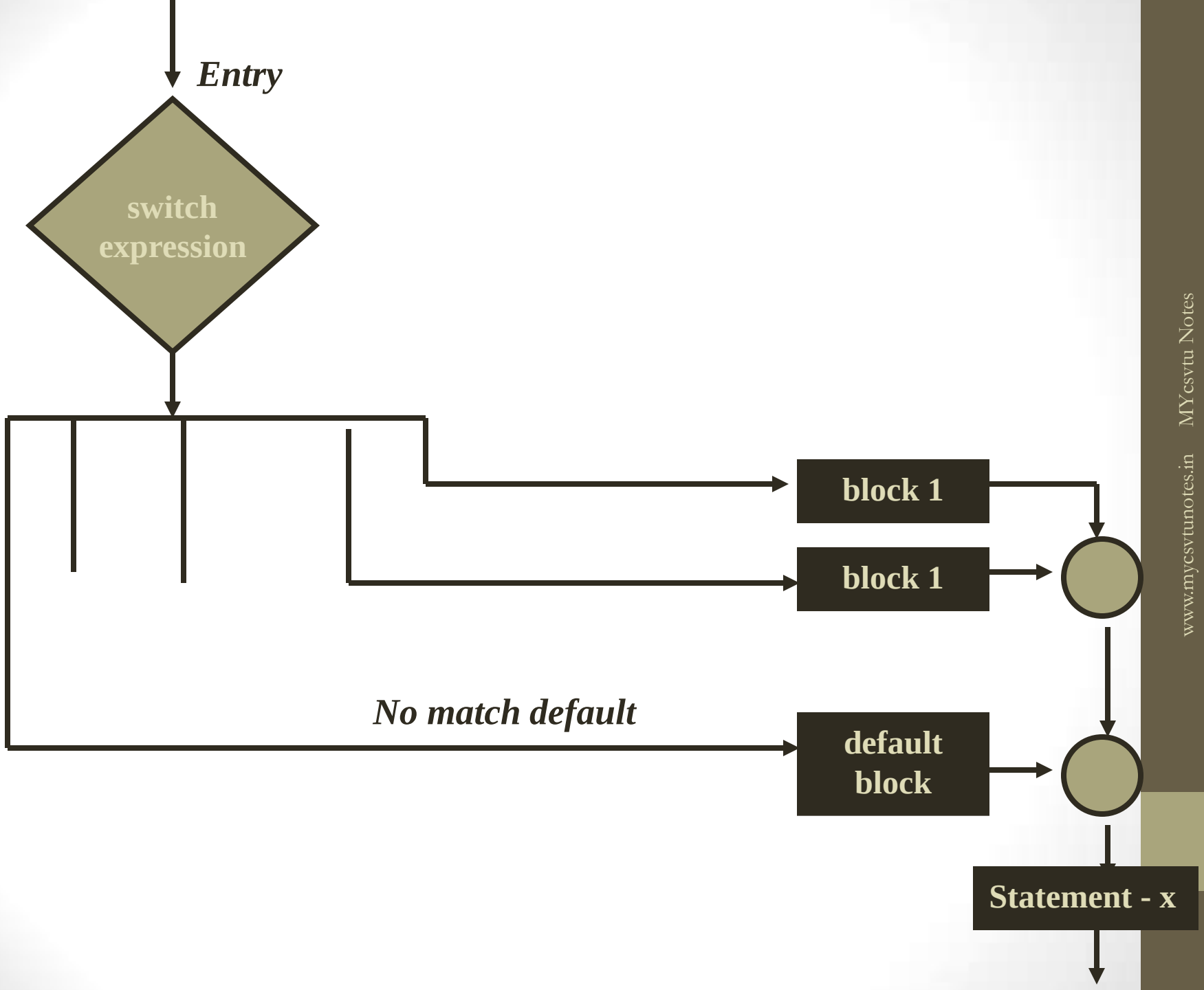
This is used when multipath decisions are involved. A multipath decision is a chain of ifs in which the statement associated with each else is an if.





switch statement

```
switch(expression)
{
    case value-1:
                block-1
                break;
     case value-2:
                block-2
                break;
       ……….
       ……….
     default :
            default block
            break;
}
```



? : operator

The C language has an unusual operator, useful for making two-way decisions. This operator is a combination of ? And : and takes three operands. This operator is popularly known as conditional operator.

The general form of conditional operator is :

conditional expression ? expression1 : expression2

The conditional expression is evaluated first. If the result is nonzero, expression1 is evaluated and is returned as the value of the conditional expression. Otherwise, expression2 is evaluated and its value is returned.

$$\text{Salary} = \begin{cases} 4x + 100 & \text{for } x < 40 \\ 300 & \text{for } x = 40 \\ 4.5x + 150 & \text{for } x > 40 \end{cases}$$

Can be written as :

Salary=(x<40) ? (4*x+100):(x>40)?(4.5x+150):300;

LECTURE 11

goto STATEMENT

goto statement

C supports the goto statement to branch unconditionally from one point to another in the program.

The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name, and must be followed by a colon. The label is placed immediately before the statement where the control is to be transferred.

```
goto label;
…….
…….
label:
statement;
```

```
label:
statement;
…….
…….
goto label;
```

The label can be anywhere in the program either before or after the goto label: statement.

If the label is before the statement goto label; a loop will be formed and some statement will be executed repeatedly. Such a jump is known as a backward jump.

If the label: is placed after the goto label; some statements will be skipped and the jump is known as a forward jump.

LECTURE 12

LOOPING

LOOPING

In looping a sequence of statements are executed until some conditions for the termination of the loop are satisfied.

A program loop consist of two segments , one known as body of loop and the other known as control statement .

The control statement test certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

A looping process includes the following steps :

- Setting and initialization of a condition variable
- Execution of the statements in the loop
- Test for a specified value of the condition variable for the execution of the loop
- Increment or updating the condition variable

The test may be either to determine whether the loop has been repeated the specified number of times or to determine whether a particular condition has been met.

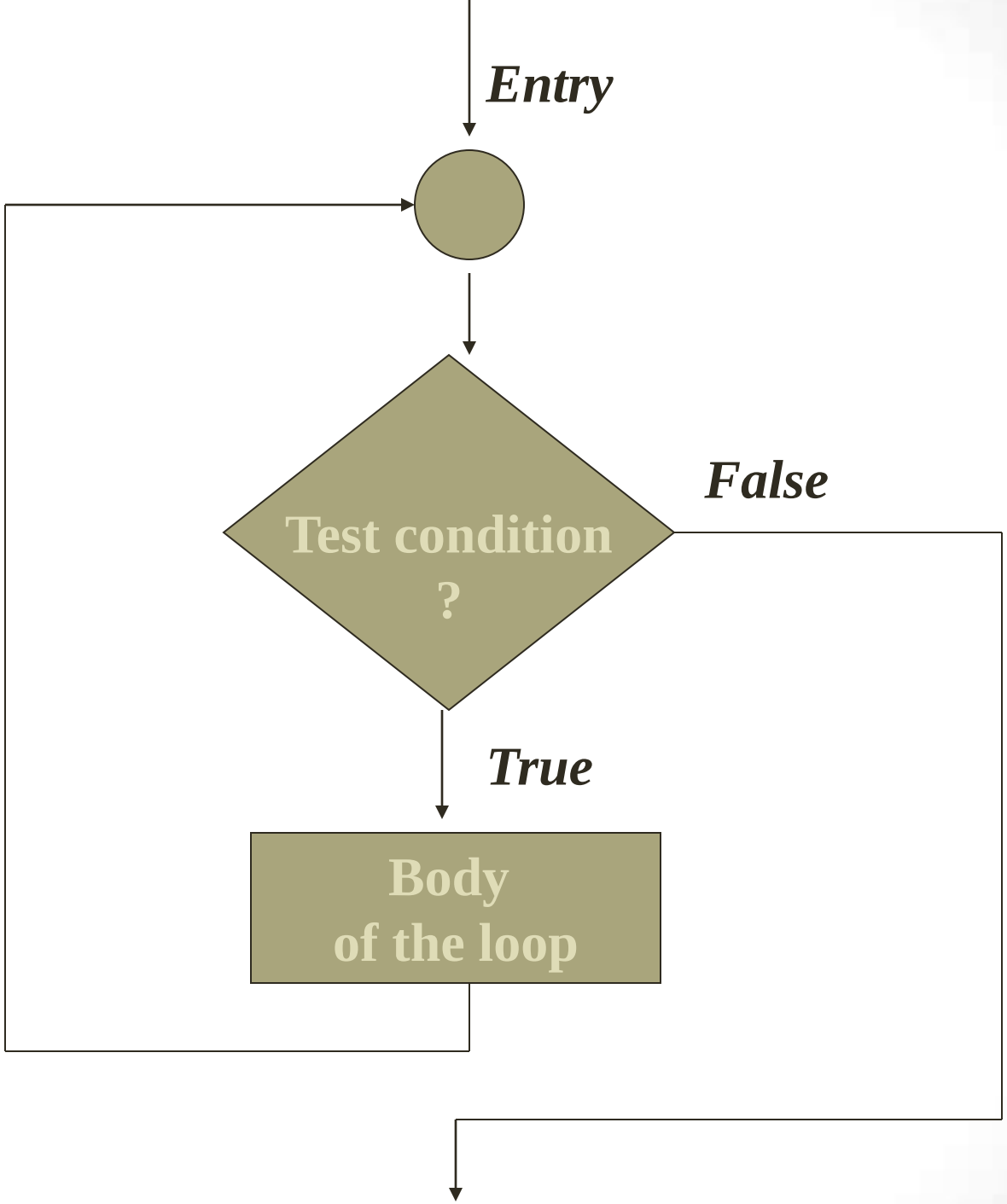
Depending on the position of the control statement in the loop, a control structure may be classified as:

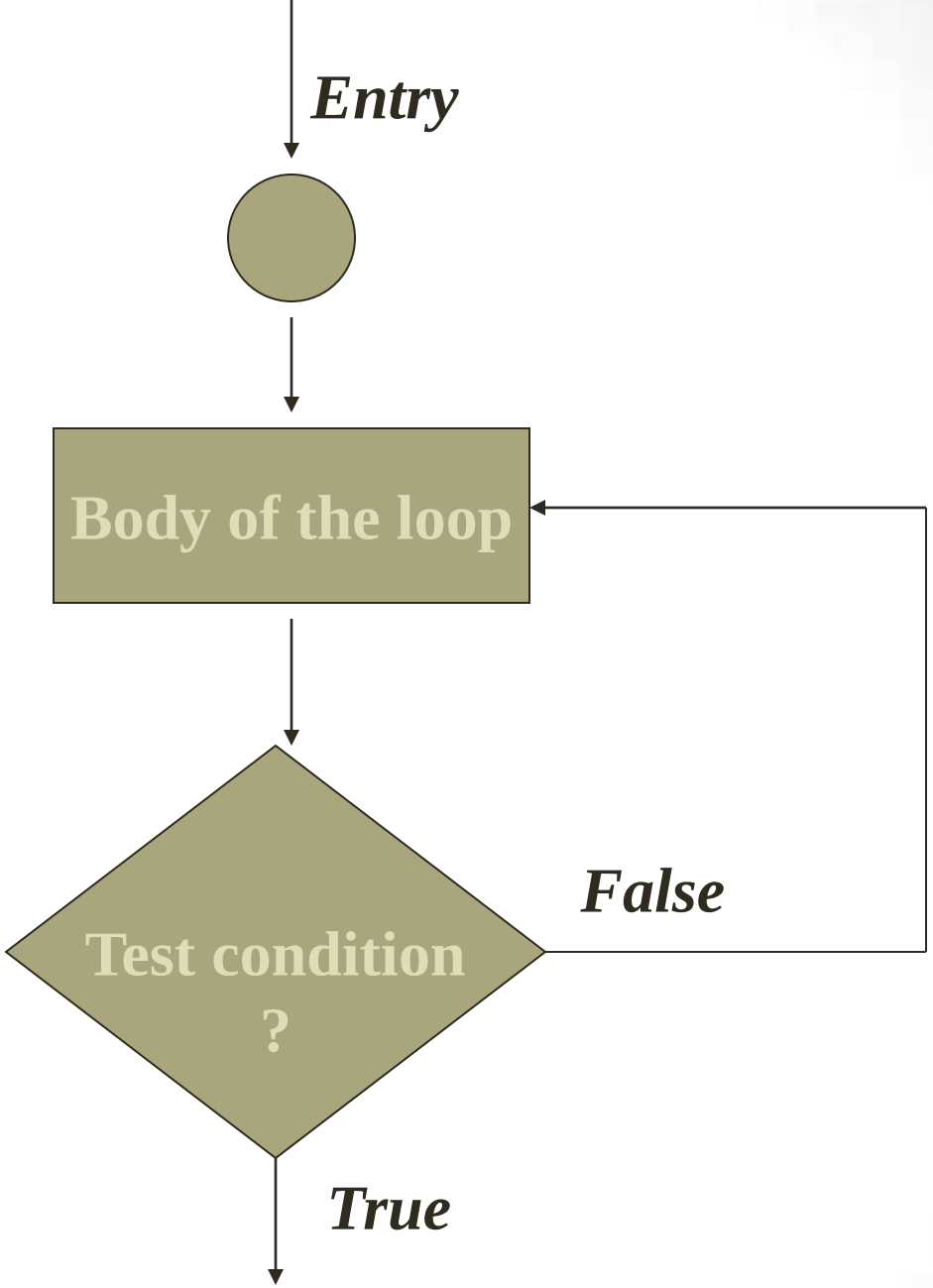
- ✓ **Entry-controlled loop :**

  Control conditions are tested before the start of the loop execution

- ✓ **Exit-controlled loop :**

  Control conditions are tested at the end of the body of the loop





Based on the nature of control variable and the kind of value assigned to it for testing the loop may be classified as :

1. Counter-controlled loops or Definite repetition loop :

It is used when we know in advance exactly how many times the loop will be executed

Control variable is known as counter.

For example : 10, 20

2. Sentinel-controlled loops or Indefinite repetition loop :

In this type of loop a special value called a sentinel value is used to change the loop expression from true to false

Control variable is known as sentinel variable

For example –1 and 999

while statement

```
while(test condition)
{
    body of loop
}
```

For example :

```
n=1;
while(n>=10)
{
    printf(“%d”,n);
    n=n+1;
}
```

do statement

```
do
 {
    body of loop
 }  while(test condition);
```

For example :

```
n=1;
 do
{
    printf("%d",n);
    n=n+1;
}while(n>=10);
```

LECTURE 13

for loop

```
for( initialization  ; test-condition  ; increment)
{

   body of loop

}
```

For example :

```
for( n = 0; n < = 10; n + +)
{
    printf(“%d” , n);
}
```

The `for` Repetition Structure

- For loops can usually be rewritten as while loops:

```
initialization;
while ( loopContinuationTest ) {
  statement;
  increment;
}
```

- Initialization and increment
  - Can be comma-separated lists
  - Example:

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++)
   printf( "%d\n", j + i );
```

The `for` Structure: Notes and Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions. If **x** equals **2** and **y** equals **10**

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

  is equivalent to

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

- Notes about the **for** structure:
  - "Increment" may be negative (decrement)
  - If the loop continuation condition is initially **false**
    - The body of the **for** structure is not performed
    - Control proceeds with the next statement after the **for** structure
  - Control variable
    - Often printed or used inside for body, but not necessary

The `break` and `continue` Statements

- **`break`**
  - 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

The `break` and `continue` Statements

- **`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`**
    - Increment expression is executed, then the loop-continuation test is evaluated

```c
/* Fig. 4.12: fig04_12.c
   Using the continue statement in a for structure */
#include <stdio.h>

int main()
{
   int x;

   for ( x = 1; x <= 10; x++ ) {

      if ( x == 5 )
         continue;  /* skip remaining code in loop only
                       if x == 5 */

      printf( "%d ", x );
   }

   printf( "\nUsed continue to skip printing the value 5\n" );
   return 0;
}
```

1. Initialize variable

2. Loop

3. Print

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

Program Output