History of ANSI C

- ALGOL: Root of all modern languages introduced in 1960s. Used Block structures
- 1967 Martin Richard developed Basic Combined Programming Language (BCPL).
- 1970 Ken Thompson created a language using BCPL called B.
- Evolution over the years:
- 1972 – C invented by Dennis Ritchie at Bell Labs.
- 1978 – *The C Programming Language* published; first specification of language
- 1989 – C89 standard (known as ANSI C or Standard C)
- 1990 – ANSI C adopted by ISO, known as C90
- 1999 – **C99** standard
  - Mostly backward-compatible
- 2007 – work on new C standard C1X announced
- C11 was officially ratified and published on December 8, 2011. Features included **Atomic Operations, Thread Support, Static Assertions.**
- October 2018, "C17" is the current **standard.** Feature include **Added Standard Headers, improved Unicode, New Character function.**
- **C23** is the informal name, what will likely become **ISO/IEC 9899:2024**, will replace **C17** (standard ISO/IEC 9899:2018)

Importance of C

- **Foundational Language**. Most language finds its root in C with syntax and concepts of other languages such as C++, Java, and even Python finding analogy with C.
- **Efficiency and Performance:** Low-level programming language allowing direct memory manipulation and offering high performance. C compiler combines the capabilities of an assembly language with features of a high-level language. Suited for writing system software and business packages.
- **Portability:** Ported across different platforms and architectures. Standardized libraries and syntax helps in separating language from machine-specific details. Can be used in different
- **Extensive standard Library:** Rich standard library with large number of built-in functions and utilities
- **Wide application Range:**
- **Interface with Hardware:** Ability to perform memory manipulation and interact directly with hardware makes it well-suited for embedded systems and device driver development.
- **Language for Learning Programming Concepts:** Often used for teaching programming concepts and data structures. Helps learners understand fundamental programming concepts without the complexities of higher-level languages.
- **Compatibility with C++**: C++ is extension of C. Basic understanding of C helps transition from C++

Keywords of C

- 32 keywords in C.
- 27 keywords given by Ritchie.
- 5 added by ANSI

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

Basic Structure of C

- C program viewed as a group of building blocks called **functions**.
- **Function** is a subroutine containing more than one ***statements*** used to perform a ***specific tasks***.
- ***Documentation*** section consists of a set of comment lines giving the details of the program.
- ***Link*** section provides instructions to the compiler to link functions from system library.
- ***Definition*** section defines all symbolic contants.
- ***Global variables*** used in more than one functions. Declared in *global declaration* section.

Basic Structure of C contd.

C program is a collection of one or more functions. Every function is a collection of statements an performs some specific task. The general structure of C program is-

```
Comments
Preprocessor directives
Global variables
main( ) function
{
    local variables
    statements
    ……..
    ……..
}
func1( )
{
    local variables
    statements
    ……..
    ……..
}
func2( )
{
    local variables
    statements
    ……..
    ……..
}
```

Comments can be placed anywhere in a program and are enclosed between the delimiters /* ar */. Comments are generally used for documentation purposes.

Programming Style

- C is a ***free-form*** language, i.e., compiler does not care where on the line we begin typing.
- ***C program statements written in lowercase letters***.
- ***Uppercase letters*** used for symbol constants. e.g. #define PI 3.1416
- ***Braces, group program statements*** mark the beginning and end of a function. Proper indentation increase readability of program.
- ***Statements*** can be grouped together on one line.

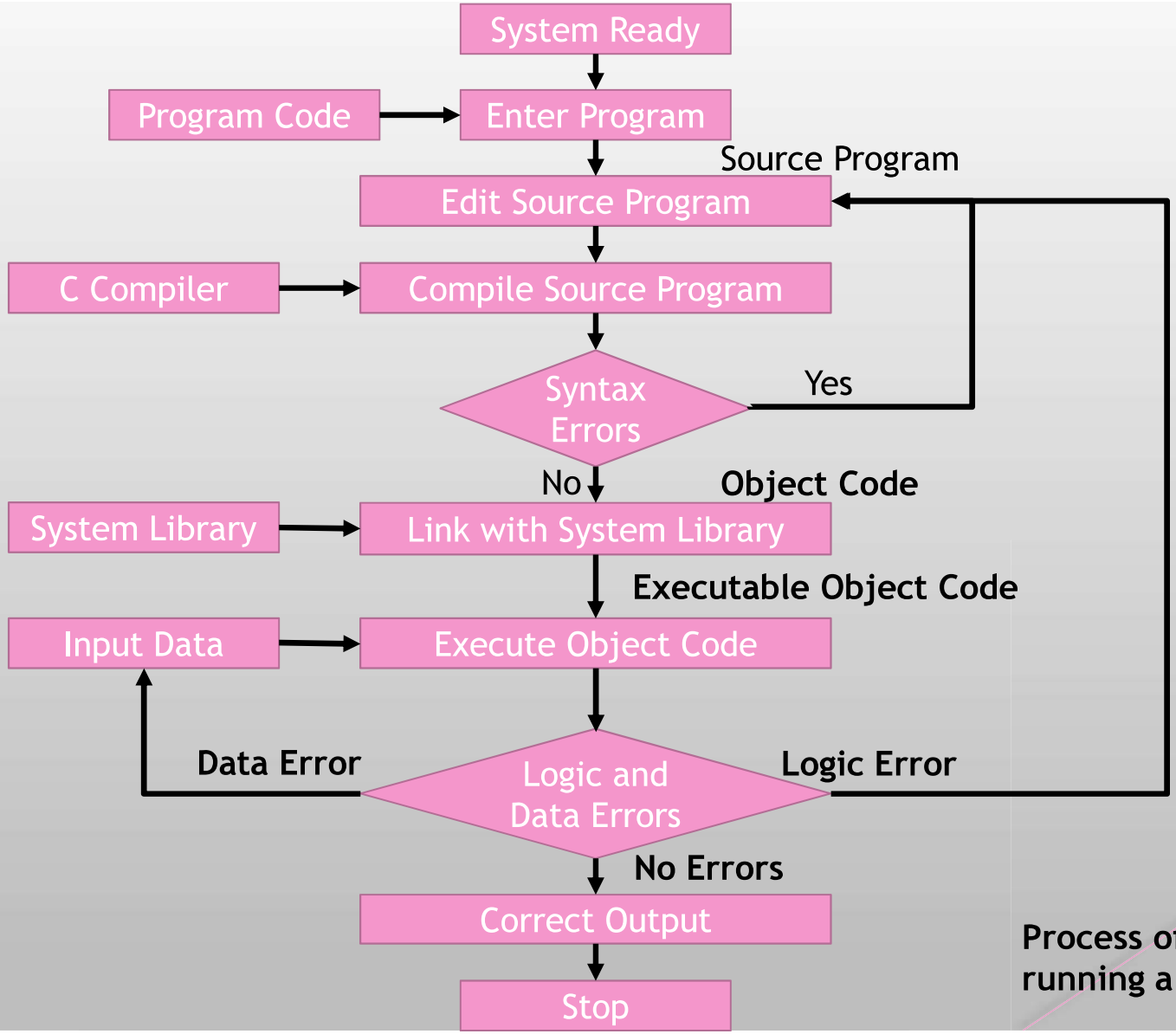
```
x = 1;
y = 2;
a = 3;
z = x + y;
p = a - x;
```

```
x = 1; y = 2; a = 3; z = x + y; p = a - x;
```

- ***Comments*** increase code readability and also helps in understanding the logic.

Executing C Program

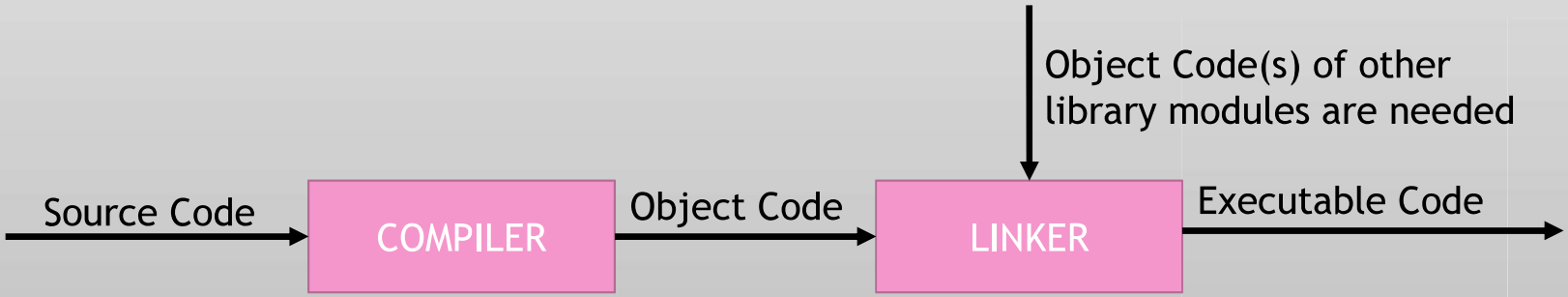
- Steps involved in execution of C program.
  1. Program Creation.
  2. Program Compilation
  3. Linking Program with functions needed from C library
  4. Program Execution
- Process of ***Creating, Compiling, and Executing*** remain same irrespective of the ***operating system (OS)***.
- OS is a program that controls entire operation of a computer system. I/O operations channeled through OS, and act as interface between hardware and the user.
- Most popular OS are UNIX (for minicomputers) and MS-DOS (for microcomputers)



Process of Compiling and running a C program

Executing C Program contd.

- ***Source Code*** is file that contains the C source program.
- ***Compiler*** translates the source program to ***object code***.
- ***Object code*** contains set of machine instructions which acts as an input to computer.
- A given C program use **several library functions**.
- ***Linker*** is system software that links together all the object codes to generate the final ***executable code***.



Generation of Executable Code From Source Code

UNIX System

- **Program Creation:** After loading UNIX OS into memory, program ready to receive program. Program must be entered into a file. Filename consists of alphabets, digits, and special characters, followed by a dot (.) and letter **c.** *e.g.*

  Welcome.c

  testprogram.c

  trial_01.c

  - File is created with the help of text editor, either **ed** or **vi.**
  - Corrections in the program are done under editor. Program entered into the file is called *source program*.

- **Program Compilation and Linking:**

  - Let us assume that the source program ***gs1.c*** has been created and is ready for compilation. In UNIX compilation is accomplished by

    cc *gs1.c*

  - Instructions in source program are translated after examining each instruction by the ***Compiler***. Correct compilation leads to creation of object code file with name ***gs1.o***
  - ***Linking*** process combine together other library files or functions required to run the program. If program use ***exp()*** function, then object code of this function should be brought from ***math library***. Compiled and linked program is called ***executable object code*** and stored automatically as ***b.out***.

UNIX System contd.

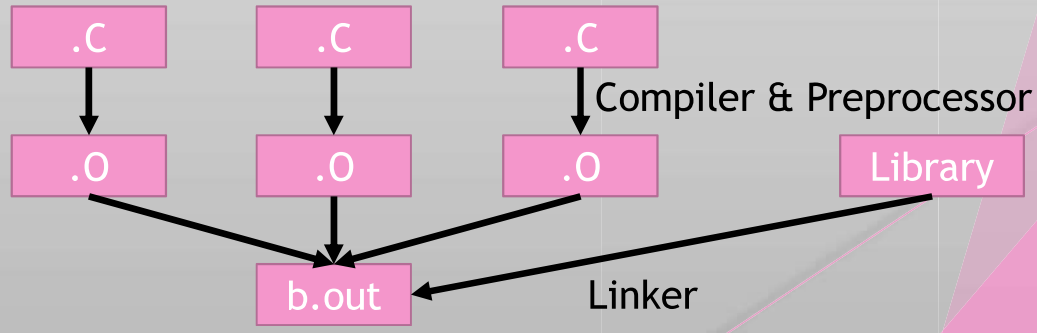
- **Program Execution:** Program execution is easy and begin by executing command

  ./b.out

  - Command would load the executable object code into the computer memory and execute the instructions. During execution, the program may require data input. Modification in source program leads to repetition of compiling, linking, and executing process.
  - **Creation of Own Executable file:** Linker assigns the name **b.out** which is overwritten by executable object code of new program. File Renaming is necessary to prevent overwriting, and is accomplished by command

    mv b.out *name*

  - **Multiple Source File:** Append multiple filenames to **cc** command for compilation and linking of multiple files.



MS-DOS System

- **Program Creation** using word processing software in non-document mode. e.g.

trial.c

- Then command

MSC trial.c

load the program stored in file **trial.c** and generate **object code**.

- **trial.obj** is the object file. Debug for any error and recompile again.
- Linking done by command

LINK trial.obj

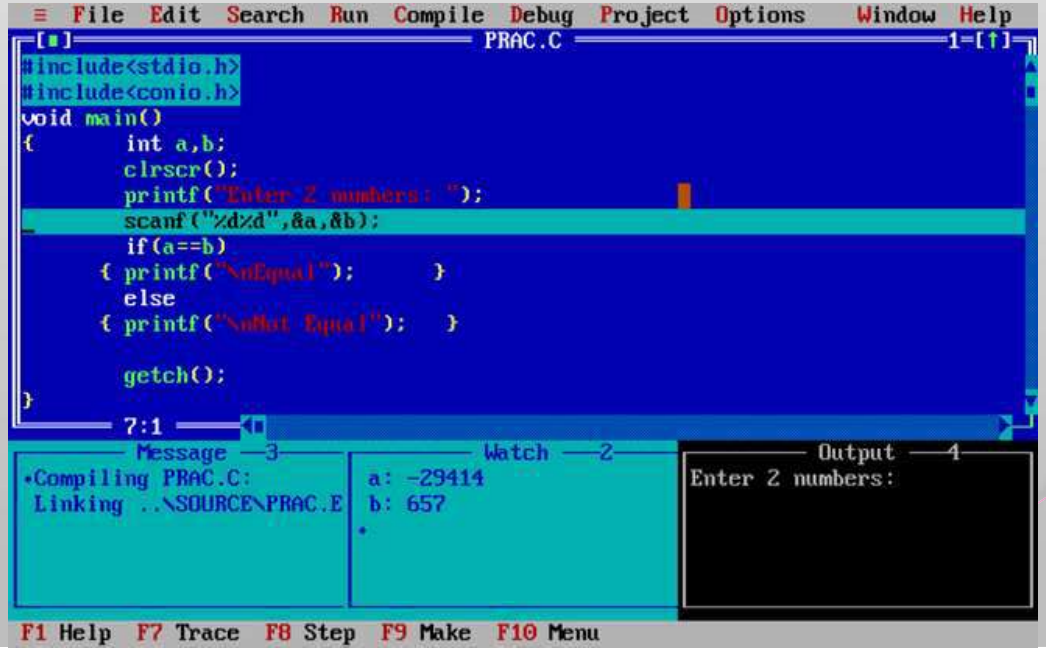
- After linking the **executable code** with filename **trial.exe**
- Command

trial

- Would execute the program

Windows System

- Unlike LINUX windows system does not come with an in-built C compiler.
- Several Integrated Development Environment (IDE) are utilized to write, save, compile, edit, link and run programs.
- IDEs needs to be installed separately on a Windows system.
- Examples: **Borland C/C++, Microsoft Visual studio, Dev-C++, Turbo C/C++.**
- IDE's have extremely friendly graphic user interface.



Common Programming Errors

- C does not check and report run time errors.
- Commonly programming errors experienced during C program are

1. **Missing Semicolon**: Every C statement must end with a semicolon. Missing semicolon results in 'misleading' error messages. e.g.

```
s=a+b
d=x/y;
```

- Compiler will treat the second line as a part of the first one and treat **d** as a variable name.
- We get "**undefined name**" error message after compilation.
- ***NOTE: Both the message and location are incorrect.***

2. **Misuse of Semicolon:** Putting semicolon in wrong place. e.g.

```
for(i=1;i<=10;i++);
sum=sum+i;
```

- Code is supposed to sum all the integers from 1 to 10. Although code is syntactically correct, however only the last value of **i** which is **10** is added and the output is **10**.
- Few more examples are

```
while(x<Max);
{
}
if(Totalmarks>=90);
grade = 'A';
```

Common Programming Errors contd.

3. **Use of = Instead of ==.** While performing **relational test** always use **comparison operator** and **not the assignment operator**. e.g.

```
if (sum = 1)
count++;
```

- This is a syntactically valid statement. The variable **sum** is assigned 1 and then count is incremented. The statement does not perform any relational operation on the code. Irrespective of the previous value of code, **count++;** is always executed.
- Similar mistakes can occur in looping statements, i.e., **for** and **while.** Mistake in looping control statements might cause **infinite loops.**

```
#include <stdio.h>
int main()
{
int i=1;
while(i<=10);
{
printf("%d", i);
i++;
}
return 0;
}
```

Common Programming Errors contd.

4. **Missing Braces:** In nested loop the number of opening and closing braces should match with the closing ones. Further, placing/omitting the matching braces at wrong location would results in unexpected results. e.g.

```
for(i=1; i<= 10; i++)
sum1 = sum1 + i;
sum2 = sum2+ i*i;
printf("%d %d\n", sum1,sum2);
```

- Code intends to compute **sum1**, **sum2** for i varying from 1 to 10, in steps of 1 and print their values. But the compiler interprets the code as

```
for(i=1; i<= 10; i++)
{
sum1 = sum1 + i;
}
sum2 = sum2+ i*i;
printf("%d %d\n", sum1,sum2);
```

- **sum1** is treated inside the loop while **sum2** is executed once the execution of looping statement ends.

Common Programming Errors contd.

5. **Misusing Quotes:** Single quotes are used while handling characters. Care should be exercised while declaring variables. e.g. **city = 'A';** becomes invalid when **city** is declared as **char** variable with dimension.
6. **Improper Comment Character:** Comments should begin with **/*** and end with ***/**. Missing ***/** results in confusion to C compiler, and would result in error. ***Remember: C does not support nested comments.***

```
. . . . . .
/*Program for printing the Hello
printf("Hello\n");
/* Comment it */
. . . . . .
```

7. **Undeclared Variables**
8. **Forgetting the Precedence of Operators:** Expressions are evaluated according to the precedence of operators. e.g.

- Logically incorrect way

```
if (value = mul ()>=100)
tax = 0.05 * value;
```

- Logically correct way

```
if ((value = mul ())>=100)
tax = 0.05 * value;
```

Common Programming Errors contd.

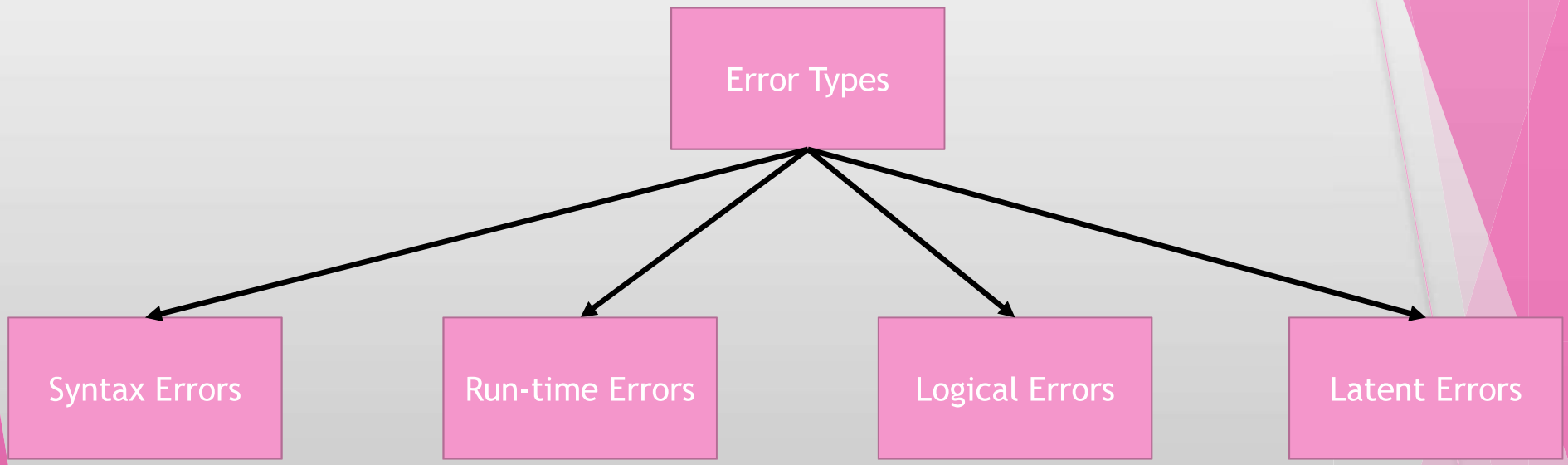
9. **Ignoring the order of Evaluation of Increment/Decrement Operators:**

```
. . . . . .
i =0;
while ((c = getchar()) != '\n')
{
string[i++] = c;
}
string[i-1] = '\n';
```

- Statement **string[i++] = c;** is equivalent to: **string[i] = c; i = i+1;** this is not same as **string[++i] = c;** which denote i=i+1; string[i] = c;

10. **Forgetting to Declare Function Parameters**
11. **Forgetting a Space for Null character in a String**
12. **Using Uninitialized Pointers**
13. **Missing Parentheses in Pointer Expression**
14. **Missing Indirection and Address Operators**

Types of Errors



Program Testing

- Process of reviewing and executing a program with the intent of detecting error.
- **Compiler can detect syntactic and semantic error**.
- Testing process may include following two stages:
  - **Human Testing**
  - **Computer-based testing**
- ***Human Testing*** an **effective error-detection process** and is done before the computer-based testing. Human testing **comprise of code inspection by programmer, test group, and reviewer.**
- ***Computer-based testing*** involves two stages, namely ***compiler testing and run-time testing***.
- ***Compiler testing*** is simplest and detects for undiscovered syntax errors.
- ***Run-time testing*** rectifies run-time error messages such as "**null pointer assignment**" and "stack overflow".
- Program testing done at ***module (function) level*** or at ***program level***.
- Module level test known as ***unit test***, conducted on each module to uncover errors within the boundary of the module.
- Different modules are integrated and finally an integration test is performed to discover error associated with interfacing.

Program Debugging

- Process of isolating and correcting errors.
- One method is to place print statements throughout the program to display variable values.
- Displays dynamics of a program and allow programmer to examine and compare the information at various points.
- ***Conditional compilation statements*** to switch on or off the debugging statements.
- ***Deduction*** process is another approach to debug a program. Here the error location is obtained by using the process of elimination and refinement, and is done using list of ***possible error causes***.
- ***Third error-locating*** method is the ***backtrack approach***. The error is identified using backtrack approach where the logic of program is revisited to locate the possibility of mistake, i.e., beginning at the place where symptom is discovered, the program is traced backward until the error is located.