.1 PROBLEM SOLVING USING COMPUTERS

efinition: Solving any sort of problem is the basic aim of the mputer. The person who writes a solution to the problem is ogrammer.

ogrammers understand how a human can solves a problem, en translates this "algorithm" into a bit of computer can do, and ally "write" the detailed syntax (requisite by a computer) to mplete the task.

metimes, it is the case that machine tries to implement in fferent way than any human can do.

mputer Programmers are basically problem solvers. In quence to solve a problem on a computer you must:

Know how to represent the information (data) describing the problem.

Determine the steps to transform the information from one representation into another.

.2 INTRODUCTION TO COMPUTER PROGRAMMING

efinition: Programming is a set of specific instructions which e given to computer for execution of certain process or for rtain problem solving.

These instructions can be precise in one or more programming languages like C, C++, Java and so on.

A computer follows a set of steps whose intention behind is to achieve something.

These steps are instructed to the computer by computer programs.

Basically, computer programming is the process by which these programs are designed and implemented.

Computer Programming can be done with the help of various programming language.

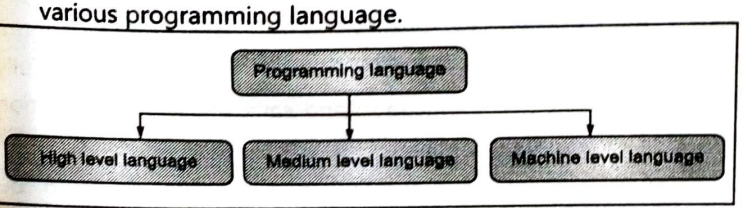


Fig. 1.1 :Distribution of Programming Language

High Level Language (HLL)

Definition: A HLL is a programming language which enables a normal human being like us to write programs that helps the computer to perform any of the required/ instructed task.

- These types of Programming Language are easy to understand, read and write.
- But this type of Programming Language is never ever understood by the computer.
- To make our computer understand that language we require converting it to Machine Language.
- Before converting it to any machine level the high level language gets converted to Assembly level Language.

Medium Level Language

Definition: Medium-level language (MLL) is a computer programming language that interacts with the abstraction layer of a computer system.

- Middle level language is also known as Assembly level language / intermediate programming language.
- It is basically the output of programming code which is written in high level language.
- Before executing the actual code written in HLL , it needs to translated so that the processor understands it.

C intermediate language and Java byte code are some examples of medium-level language.

Low Level Language:

Definition: It is a programming language that deals with a computer's hardware components and constraints.

- It is the only language which is only and only understood by the computer and no human can actually read or understand it.
- So these languages are always considered to be closer to computers.
- Low level languages are also known as Machine Level Languages.

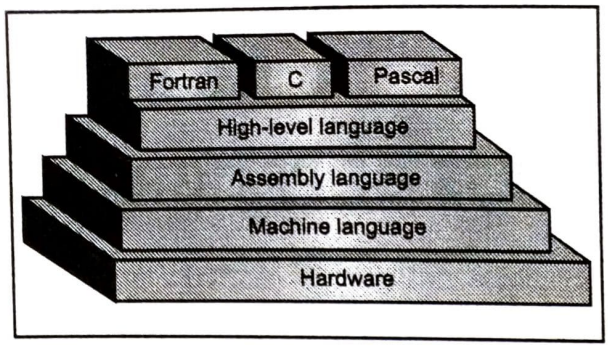


Fig. 1.2 : Levels of Programming Language

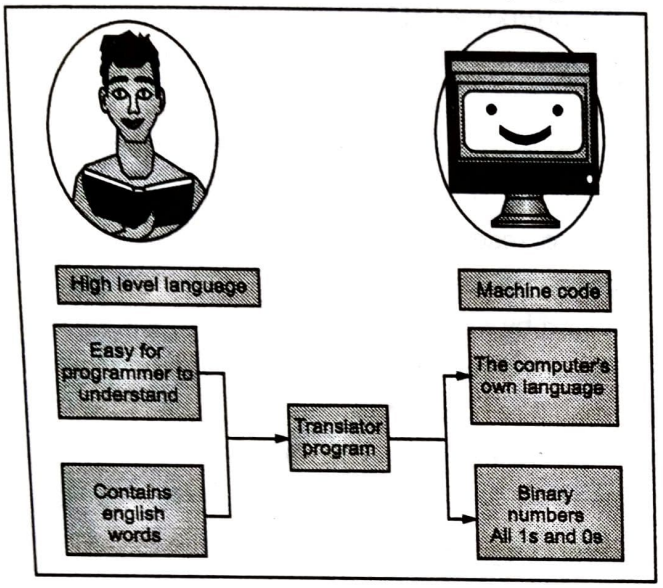


Fig. 1.3 : High Level Programming Lang. to Machine Code

1.3 EDITING THE PROGRAM

Developing a program in a compiled language such as C requires at least four steps:

1. Editing (or writing) the program
2. Compiling it
3. Linking it
4. Executing it

We will now cover each step separately.

1. Editing

You write a computer program with words and symbols that are understandable to human beings. This is the editing part of the development cycle. You type the program directly into a window on the screen and save the resulting text as a separate file. This is often referred to as the source file (you can read it with the TYPE command in DOS or the cat command in unix). The custom is that the text of a C program is stored in a file with the extension .c for C programming language

2. Compiling

You cannot directly execute the source file. To run on any computer system, the source file must be translated into binary numbers understandable to the computer's Central Processing Unit (for example, the 80*87 microprocessor). This process produces an intermediate object file - with the extension .obj, the .obj stands for Object.

3. Linking

The first question that comes to most peoples minds is Why i linking necessary? The main reason is that many compile languages come with library routines which can be added to you program. Theses routines are written by the manufacturer of the compiler to perform a variety of tasks, from input/output to complicated mathematical functions. In the case of C the standard input and output functions are contained in a library (stdio.h) so even the most basic program will require a library function. After linking the file extension is .exe which are executable files.

4. Executable files

Thus the text editor produces .c source files, which go to the compiler, which produces .obj object files, which go to the linker, which produces .exe executable file. You can then run .exe files as you can other applications, simply by typing their names at the DOS prompt or run using windows menu.

1.4 COMPILING AND EXECUTING C PROGRAM

- C is compiled language.
- Once the program is written, you must run it through C compiler that can create an executable file to be run by the computer.While the C Program is human–readable on other side executable file is machine readable.
- The compiler translates source code into an object code.The object code contains machine instructions for the CPU.
- However object file is not executable file.Therefore in next step,the object file is processed with another special program called as Linker.The output of the linker is an executable or runnable file.The process is shown in Fig. 1.4.

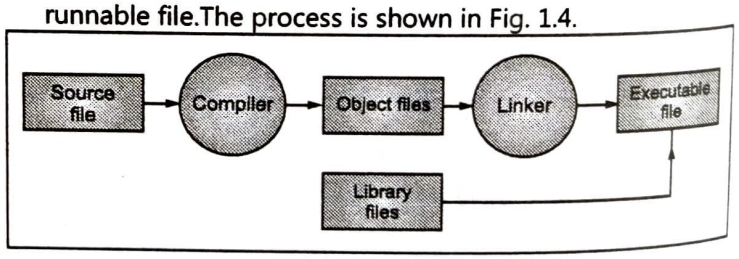


Fig. 1.4 : Overview of Compilation and Execution Process

1.5 ERROR CHECKING

Error handling refers to the anticipation, detection, and resolution of programming, application, and communications errors. Specialized programs, called error handlers, are available for some applications. The best programs of this type forestall errors if possible, recover from them when they occur without terminating the application, or (if all else fails) gracefully terminate an affected application and save the error information to a log file.

In programming, a development error is one that can be prevented. Such an error can occur in syntax or logic. Syntax errors, which are typographical mistakes or improper use of

special characters, are handled by rigorous proofreading. Logic errors, also called bugs, occur when executed code does not produce the expected or desired result. Logic errors are best handled by meticulous program debugging. This can be an ongoing process that involves, in addition to the traditional debugging routine, beta testing prior to official release and customer feedback after official release.

A run-time error takes place during the execution of a program, and usually happens because of adverse system parameters or invalid input data. An example is the lack of sufficient memory to run an application or a memory conflict with another program.

As such, C programming does not provide direct support for error handling but being a system programming language, it provides you access at lower level in the form of return values. Most of the C or even Unix function calls return -1 or NULL in case of any error and set an error code errno. It is set as a global variable and indicates an error occurred during any function call. You can find various error codes defined in <error.h> header file.

So a C programmer can check the returned values and can take appropriate action depending on the return value. It is a good practice, to set errno to 0 at the time of initializing a program. A value of 0 indicates that there is no error in the program.

errno, perror(), and strerror()

The C programming language provides perror() and strerror() functions which can be used to display the text message associated with errno.

The perror() function displays the string you pass to it, followed by a colon, a space, and then the textual representation of the current errno value.

The strerror() function, which returns a pointer to the textual representation of the current errno value.

Let's try to simulate an error condition and try to open a file which does not exist. Here I'm using both the functions to show the usage, but you can use one or more ways of printing your errors. Second important point to note is that you should use stderr file stream to output all the errors.

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

externinterrno ;

int main () {

  FILE * pf;
interrnum;
pf = fopen ("unexist.txt", "rb");

if (pf == NULL) {

errnum = errno;
fprintf(stderr, "Value of errno: %d\n", errno);
perror("Error printed by perror");
fprintf(stderr, "Error opening file: %s\n", strerror( errnum
));
  }
else {

fclose (pf);
  }

return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Value of errno: 2

Error printed by perror: No such file or directory

Error opening file: No such file or directory

1.5.1 Divide by Zero Errors

It is a common problem that at the time of dividing any number programmers do not check if a divisor is zero and finally it creates a runtime error.

The code below fixes this by checking if the divisor is zero before dividing –

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

main() {

int dividend = 20;
int divisor = 0;
int quotient;

if( divisor == 0){
fprintf(stderr, "Division by zero! Exiting...\n");
exit(-1);
  }
```

When the above code is compiled and executed, it produces the following result –

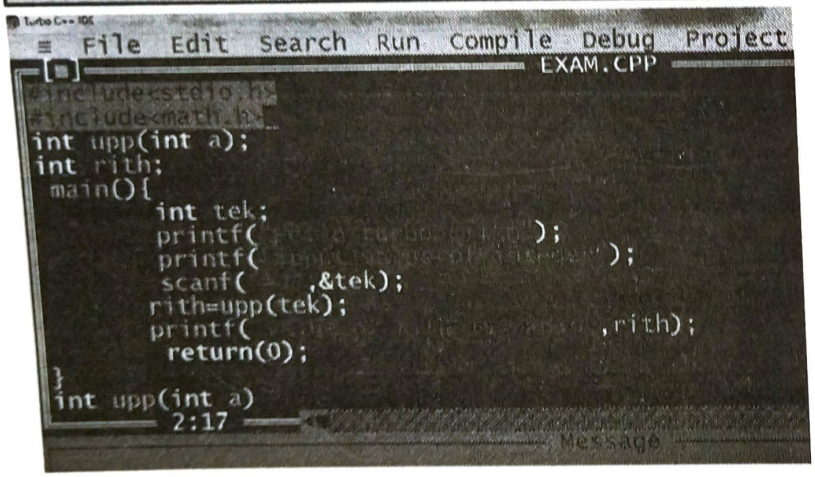
Division by zero! Exiting...

1.5.2 Program Exit Status

It is a common practice to exit with a value of EXIT_SUCCESS in case of program coming out after a successful operation. Here, EXIT_SUCCESS is a macro and it is defined as 0.

If you have an error condition in your program and you are coming out then you should exit with a status EXIT_FAILURE which is defined as -1.

1.6 EXECUTING C PROGRAMS



1.6.1 Write, Compile and Run C Program

Open a new file from File > New in the Turbo C++ IDE. Write a small program in the IDE.

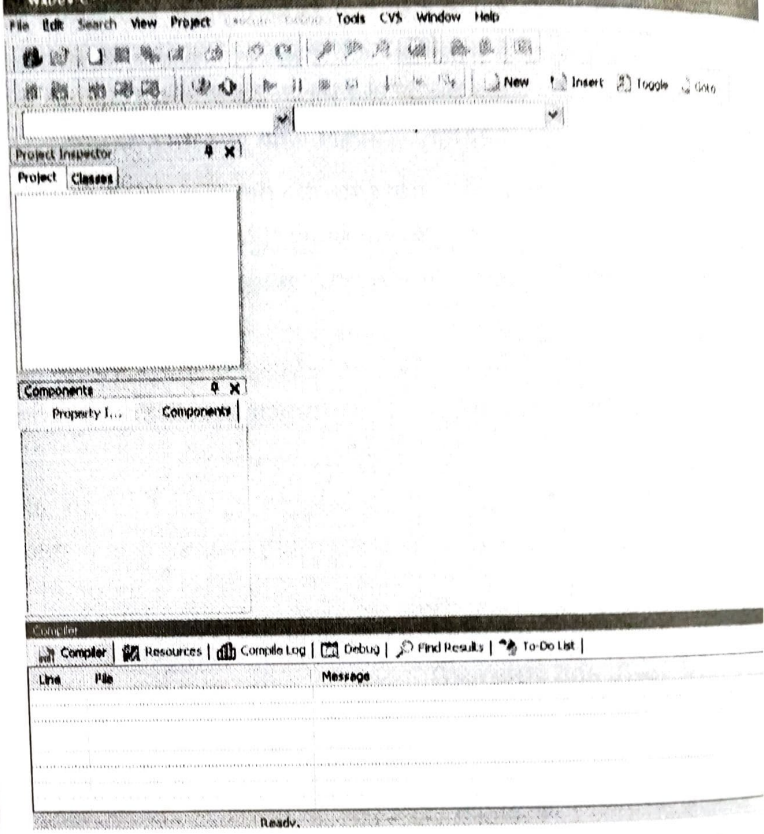
Now go to File > Save As and save the program with the filename of your choice (make sure extension of the filename is .c).

Click on Options and go to Directories. Click on Directories and set Output Directory as you want and Source Directory as where you have saved the C program file.

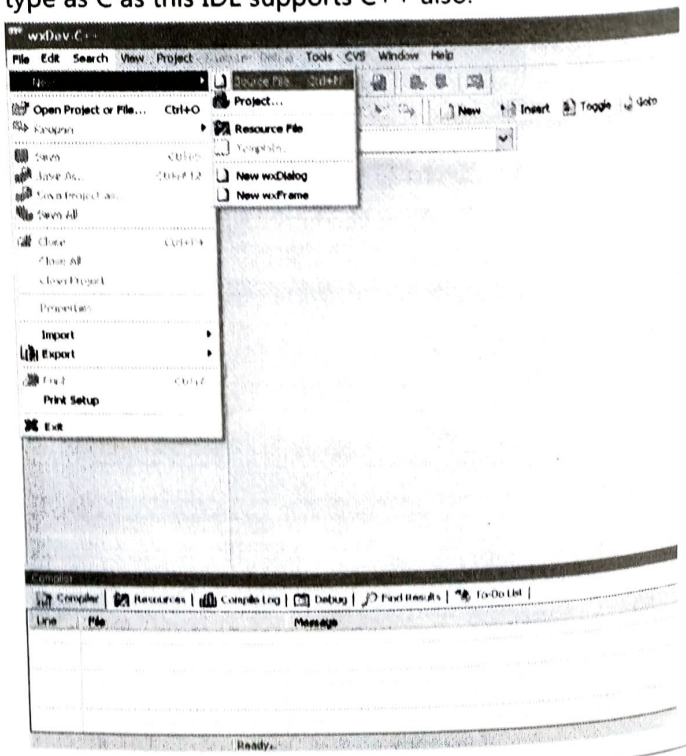
Now go to compile and click on Compile. And then Click on Run. You will see the output of your C program.

1.6.2 Write, Compile and Run C Program using wxDev-C++ in Windows

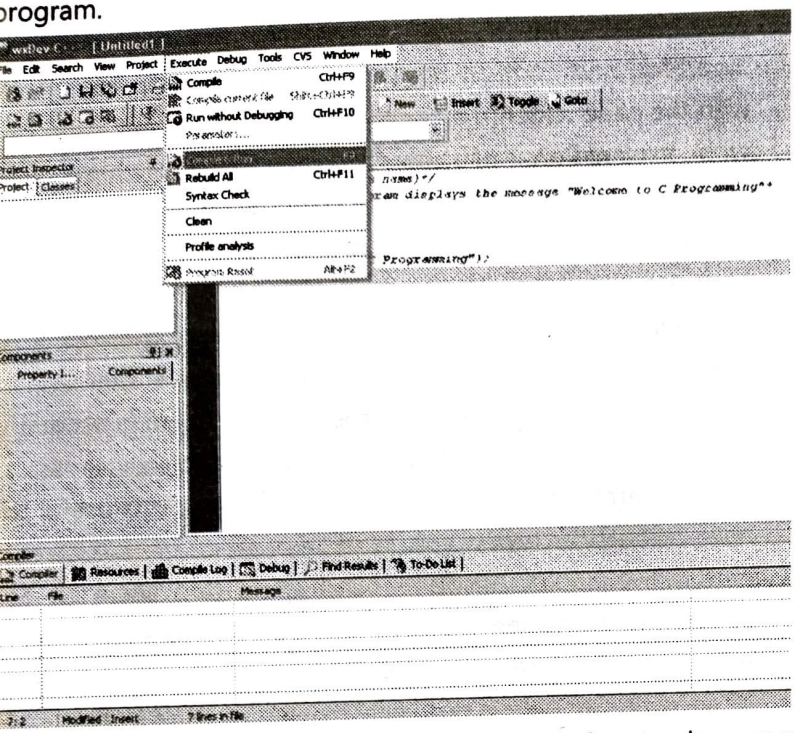
wxDev-C++ is easy to use IDE which you may opt for to write C Program. You may download the installer from wxdsgn.sourceforge.net. We found it working perfectly on Windows 7 and Windows XP. It also installs MinGW along with



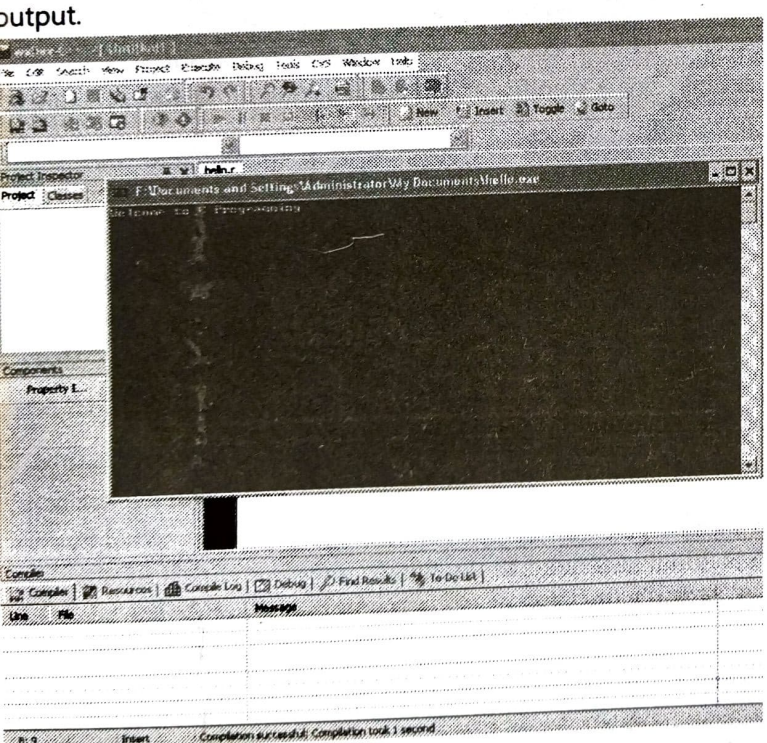
You may start programming by clicking on File > New > Sourc File in the window. Note that while saving the file, you must sele file type as C as this IDE supports C++ also.



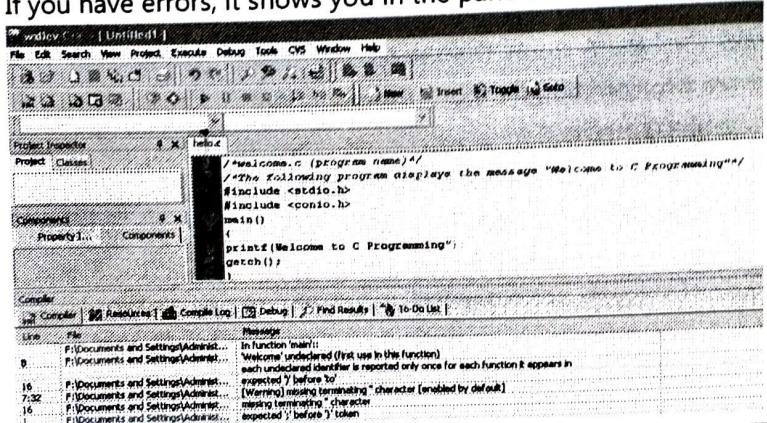
You may use F9 or as shown below to **Compile and Run** program.



When compilation is done, it opens a new window to show you output.



If you have errors, it shows you in the pane below like following.



Though slightly dated, we find wxDev-C++ an excellent IDE fo programming C. You may try it if you are using Windows.

1.6.3 Install, Compile and Execute C Program in Linux

Most of the time, when you are installing Linux, GNU Gc compiler is already installed. If not, run the following comman (our system is Ubuntu Linux) :

```
w3r@w3r-linux: ~/Desktop
File Edit View Search Terminal Help
w3r@w3r-linux:~/Desktop$ sudo apt-get install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 273 not upgraded.
w3r@w3r-linux:~/Desktop$
```

If C compiler is already installed, it will show you a message lik above. If not, it will install all the necessary packages.

Now open a text editor and write a small C program like following and save it as demo.c :

```
#include <stdio.h>

main()
{
printf("Welcome to C Programming");

}
```

Now run the command as shown below to compile and execute the file :

```
w3r@w3r-linux: ~/Desktop
File Edit View Search Terminal Help
w3r@w3r-linux:~/Desktop$ gcc -o demo demo.c
w3r@w3r-linux:~/Desktop$ ./demo
Welcome to C Programmingw3r@w3r-linux:~/Desktop$
```

This how you can install GNU Gcc compiler, write a C program and run it under Linux.

1.7 TESTING AND DEBUGGING C PROGRAM

Testing means verifying correct behavior. Testing can be done at all stages of module development: requirements analysis, interface design, algorithm design, implementation, and integration with other modules. In the following, attention will be directed at implementation testing. Implementation testing is not restricted to execution testing. An implementation can also be tested using correctness proofs, code tracing, and peer reviews, as described below.

Debugging is a cyclic activity involving execution testing and code correction. The testing that is done during debugging has a different aim than final module testing. Final module testing aims to demonstrate correctness, whereas testing during debugging is

primarily aimed at locating errors. This difference has a significant effect on the choice of testing strategies.

Preconditions for Effective Debugging

In order to avoid excessive time spent on debugging, the programmer should be mentally prepared for the effort. The following steps are useful to prepare for debugging.

- **Understand the Design and Algorithm :** If you are working on a module and you do not understand its design or its algorithms, then debugging will be very difficult. If you don't understand the design then you can't test the module because you do not know what it is supposed to do. If you don't understand the algorithms then you will find it very difficult to locate the errors that are revealed by testing. A second reason for the importance of understanding algorithms is that you may need that understanding in order to construct good test cases. This is especially true for algorithms for complex data structures.
- **Check Correctness :** There are several methods for checking correctness of an implementation prior to execution.
- **Correctness Proofs :** One useful code check is to examine code using the logical methods of correctness proofs. For example, if you know preconditions, invariants, terminating conditions, and postconditions for a loop then there are some easy checks that you can make. Does the precondition, together with any loop entry code imply that the invariant is initially true? Does the loop body preserve the invariant? Does execution of the loop body make progress towards loop termination? Does the invariant, together with the loop terminating condition and loop exit code, imply the postcondition? Even if these checks don't find all errors, you will often gain a better understanding of the algorithm by making the checks.
- **Code Tracing :** Often, errors can be detected by tracing through the execution of various calls to module services, starting with a variety of initial conditions for the module. For poorly understood psychological reasons, tracing works best if you are describing your tracing to someone else. In order to be effective, tracing of a procedure or function should be done assuming that calls to other procedures and functions work correctly, even if they are recursive calls. If you trace into a called procedure or function then you will find yourself dealing with too many levels of abstraction. This usually leads to confusion. If there is any doubt about the called procedures and functions then they can be traced separately to verify that they perform according to specifications. Again, tracing may not catch all errors, but it can enhance your understanding of algorithms.
- **Peer Reviews :** A peer review involves having a peer exa your code for errors. To be effective, the peer should already be familiar with the algorithm, or should be give algorithm and code in advance. When the reviewer with the code writer, the code writer should present the with explanations of how it correctly implements algorithm. If the reviewer doesn't understand or disa with part of the implementation, they discuss that par both are in agreement about whether or not it is an The reviewer's role is only as an aid to detecting error left to the implementor to correct them. Much of the b of a peer review derives from the psychology of prese how something works. Often the code writer discovers her own errors during the review. In any case, it is use have an outsider review your work in order to get a dif perspective and to discover blind spots that seem inherent in evaluating your own work. Like code tracing reviews can be time consuming. For class work, a peer of an entire module is not likely to pay for itself in ter instructional value. So reviews should be restricted to segments of code. For commercial programming, hov quality of the code is much more important. Thus reviews are a significant part of a software quality assu program.
- **Anticipate Errors :** Unfortunately, humans make error correctness arguments and sometimes miss cases in tracing, and peers don't always catch errors either. programmer should be prepared for some errors rem in the code after the steps listed above. Hopefully, won't be too many.

1.7.1 Requirements for Debugging

To effectively debug code you need two capabilities. Firs need to be able to efficiently call on the services provided module. Then you need to be able to get information back results of the calls, changes in the internal state of the m error conditions, and what the module was doing when an occurred.

Driving the Module

To effectively debug a module, it is necessary to have method for calling upon the services provided by the m There are two common methods for doing this.

- **Hardwired Drivers :** A hardwired driver is a main pro module that contains a fixed sequence of calls to the se provided by the module that is being tested. The seq of calls can be modified by rewriting the driver cod recompiling it. For testing modules whose behav determined by a vary small number of cases, hard drivers offer the advant... of being easy to constr

there are too many cases, though, they have the shortcoming that a considerable effort is involved in modifying the sequence of calls.

- **Command Interpreters :** A command interpreter drives the module under test by reading input and interpreting it as commands to execute calls to module services. Command interpreters can be designed so that the commands can either be entered interactively or read from a file. Interactive command interpretation is often of great value in early stages of debugging, whereas batch mode usually is better for later stages of debugging and final testing. The primary disadvantage of command interpreters is the complexity of writing one, including the possibility that a lot of time can be spent debugging the interpreter code. This is mitigated by the fact that most of the difficult code is reusable, and can be easily adapted for testing different kinds of modules. For almost all data structure modules, the flexibility offered by command interpreters makes them a preferred choice.

Obtaining Information about the Module

Being able to control the sequence of calls to module services has little value unless you can also obtain information about the effects of those calls. If the services generate output then some information is available without any further effort. However, for many modules, including data structure modules, the primary effect of calls to services is a change in the internal state of the module. This leads to needs for three kinds of information for debugging.

- **Module State :** Data structure modules generally have services for inserting and deleting data. These services almost never generate output on their own, and often do not return any information through parameters. Therefore, in order to test or debug the module, the programmer must add code that provides information about changes in the internal module state. Usually, the programmer adds procedures that can display the data contents of the module. These procedures are made available to the driver module, but are usually removed or made private when testing is complete. For debugging, it is useful to have procedures that show internal structure as well as content.
- **Module Errors :** When a module has a complex internal state, with incorrect code it is usually possible for invalid states to arise. Also, it is possible that private subroutines are called incorrectly. Both of these situations are module errors. When practical, code can be added to the module to detect these errors.
- **Execution State :** In order to locate the cause of module errors, it is necessary to know what services and private subroutines have been called when the error occurs. This is the execution state of the module. One common method for determining the execution state is the addition of debugging print statements that indicate entry and exit from segments of code.

1.7.2 Principles of Debugging

- **Report Error Conditions Immediately :** Much debugging time is spent zeroing in on the cause of errors. The earlier an error is detected, the easier it is to find the cause. If an incorrect module state is detected as soon as it arises then the cause can often be determined with minimal effort. If it is not detected until the symptoms appear in the client interface then may be difficult to narrow down the list of possible causes.
- **Maximize Useful Information and Ease of Interpretation :** It is obvious that maximizing useful information is desirable, and that it should be easy to interpret. Ease of interpretation is important in data structures. Some module errors cannot easily be detected by adding code checks because they depend on the entire structure. Thus it is important to be able to display the structure in a form that can be easily scanned for correctness.
- **Minimize Useless and Distracting Information :** Too much information can be as much of a handicap as too little. If you have to work with a printout that shows entry and exit from every procedure in a module then you will find it very difficult to find the first place where something went wrong. Ideally, module execution state reports should be issued only when an error has occurred. As a general rule, debugging information that says "the problem is here" should be preferred in favor of reports that say "the problem is not here".
- **Avoid Complex One :** Use testing code - One reason why it is counterproductive to add module correctness checks for errors that involve the entire structure is that the code to do so can be quite complex. It is very discouraging to spend several hours debugging a problem, only to find that the error was in the debugging code, not the module under test. Complex testing code is only practical if the difficult parts of the code are reusable.

1.7.3 Debugging Aids

Aids Built into Programming Language

- **Assert Statements :** Some Pascal compilers and all C compilers that meet the ANSI standard have assert procedures. The assert procedure has a single parameter, which is a Boolean expression. When a call to assert is executed the expression is evaluated. If it evaluates to true then nothing happens. If it evaluates to false then the

program terminates with an error message. The assert procedure can be used for detecting and reporting error conditions.

- **Tracebacks :** Many Pascal compilers generate code that results in tracebacks whenever a runtime error occurs. A traceback is a report of the sequence of subroutines that are currently active. Sometimes a traceback will also indicate line numbers in the active subroutines. If available, a traceback reveals where the runtime error occurred, but it is up to the programmer to determine where the cause lies.
- **General Purpose Debuggers :** Many computer systems or compilers come with debugging programs. For example, most UNIX operating systems have general purpose debuggers such as sdb and dbx. Debugging programs provide capabilities for stepping through a program line-by-line and running a program with breakpoints set by the user. When a line with a breakpoint is about to be executed the program is interrupted so that the user can examine or modify program data. Debugging programs also can provide tracebacks in case of run-time errors. Debuggers are often difficult to learn to use effectively. If they are the only tool used for debugging then it is likely that they will not save much time. For example, debugging a data structure module with a debugger, but without a good test driver, will likely result in spending a lot of time getting piecemeal information about errors.

1.7.4 Debugging Techniques

1. Incremental Testing

In a good design for a complex module, the code is broken up into numerous subroutines, most of which are no more than 10 to 15 lines long. For a module designed in this way, incremental testing offers significant advantages. For incremental testing, the subroutines are classified in levels, with the lowest level subroutines being those that do not call other subroutines. If subroutine A calls subroutine B then A is a higher level subroutine than B. The incremental testing strategy is to test the subroutines individually, working from the lowest level to higher levels. To do testing at the lower levels, the test driver must either be capable of calling the low level subroutines directly, or else the programmer must be able to provide several test input cases, each of which only involves a small number of low level subroutines. Devising these test cases requires a thorough understanding of the module algorithms, along with a good imagination. The strength of incremental testing is that at any time in the process, there are only a small number of places where errors can arise. This automatically makes debugging information more meaningful and leads to quicker determination of the cause of an error. A second reason for incremental testing is that it greatly reduces the chances of having to deal with tw more errors at the same time. Multiple errors often will gene confusing error indications.

Sanity Checks

Low level code in complex data structure is often written with assumption that the higher level code correctly implements desired algorithm. For example, the low level code may be writ with the assumption that a certain variable or parameter can be NULL. Even if that assumption is justified by the algorithm may still be a good idea to put in a test to see if the condition satisfied because the higher level code may be implemen incorrectly. This kind of check is called a sanity check. If an as procedure is available then it can be used for the checks. advantage of sanity checks is that they give early detection errors.

2. Boolean Constants For Turning Debugging Code on Off

If debugging code is added to a module then it is often profita to enclose it in an if statement that is controlled by a Boole constant added to the module. By doing this, the debugg code can easily be turned off, yet be readily available if need later. Different constants should be used for different stages testing so that useless information is minimized.

3. Error Variables For Controlling Program Behavior Af Errors

When debugging print statements are added to code, there is possibility of a tremendous explosion of useless information. problem is that a print statement by itself will be execut whether or not there is an error. Thus, if the error does appear until a large number of subroutine calls have been ma then most of the messages are just telling you everything is ok so far. This problem is greatly magnified if the added code displaying the internal structure of a data structure. Assumi that the module has sanity checks for error detection, an er Boolean variable can be added to the module. It should initialized to false, indicating that there is no error. For most da structures, there is a Create operation for initialization. The er variable can be initialized at the same time. Instead of exiting t sanity checks are modified so that they set the error variable true. Then debug code can be enclosed in if statements so th information is only printed when errors have been detected. O possible application of this method is obtaining traceba information when it is not otherwise available.

4. Traceback Techniques

To obtain a traceback, use an error Boolean set by sanity check At various places in the module add debug code controlled b the error variable that prints the current position. Usually it

more economical to first run the code with a terminating sanity check. Then you only need to add the controlled debug code at places where the subroutine that contains the sanity check is called.

5. Correcting Code Errors

For the correction of errors detected by testing, the is one very important principle to keep in mind: fix the cause, not the symptom.

Suppose that you run some code and get a segmentation fault. After some checking you determine that a NULL pointer was passed into a procedure that did not check for NULL, but tried to reference through the pointer anyway. Should you add a NULL pointer check to the procedure, enclosing the entire body of the procedure in an if statement? This question cannot be answered without an understanding of the design and algorithm. It may be that if the algorithm is correctly implemented then the pointer cannot be NULL, so the procedure does not make the check. If that is the case then adding the if statement does not fix the cause of the problem. Instead, it makes matters worse by covering up the symptoms. The problem will surely appear somewhere else, but now the symptoms will be further removed from the cause. Code such as the pointer NULL check should be added only if you are sure that it should be part of the algorithm. If you add a NULL pointer check that is not required by the algorithm then it should report an error condition. In other words, it should be a sanity check. [an error occurred while processing this directive]

1.8 IDE COMMANDS AND ECLIPSE FOR C PROGRAM DEVELOPMENT

Eclipse is an integrated development environment (IDE) for Java and other programming languages like C, C++, PHP, and Ruby etc. Development environment provided by Eclipse includes the Eclipse Java development tools (JDT) for Java, Eclipse CDT for C/C++, and Eclipse PDT for PHP, among others.

1. How to Install Eclipse C/C++ Development Tool (CDT) 8.1.2 for Eclipse 4.2.2 (Juno)

Step 1: Install MinGW GCC or Cygwin GCC

To use Eclipse for C/C++ programming, you need a C/C++ compiler. On Windows, you could install either MinGW GCC or Cygwin GCC. Choose MinGW if you are not sure, because MinGW is lighter and easier to install, but having less features.

- MinGW GCC: Read "How to Install MinGW".
- Cygwin GCC: Read "How to Install Cygwin". Make sure tha you select "gcc", "g++", "gdb", and "make" packages under the "Devel" (Development) category - these packages are not part o the default installation.

Step 2: Install Eclipse C/C++ Development Tool (CDT)

Two ways to install CDT, depending on whether you have previously installed an Eclipse:

- If you have already installed "Eclipse for Java Developers" o other Eclipse packages, you could install the CDT plug-in as follows:
  Launch Eclipse ⇒ Help ⇒ Install New Software ⇒ In "Work with" field, pull down the drop-down menu and select "Kepler - http://download.eclipse.org/releases/kepler" (o juno for Eclipse 4.2; or helios for Eclipse 3.7). In "Name" box, expand "Programming Language" node ⇒ Check "C/C++ Development Tools" ⇒ "Next" ⇒ ... ⇒ "Finish".
- If you have not install any Eclipse package, you could download "Eclipse IDE for C/C++ Developers" from http://www.eclipse.org/downloads, and unzip the downloaded file into a directory of your choice.

Step 3: Configuration

You do NOT need to do any configuration, as long as the Cygwin or MinGW binaries are included in the PATH environment variable. CDT searches the PATH to discover the C/C++ compilers.

2. Writing your First C/C++ Program in Eclipse 2.1 C++ Program

Step 1: Launch Eclipse

- Start Eclipse by running "eclipse.exe" in the Eclipse installed directory.
- Choose an appropriate directory for your workspace (i.e., where you would like to save your works).
- If the "welcome" screen shows up, close it by clicking the "close" button.

Step 2: Create a new C++ Project

For each C++ application, you need to create a project to keep all the source codes, object files, executable files, and relevant resources.

To create a new C++ project:

- Choose "File" menu ⇒ "New" ⇒ Project... ⇒ C/C++ ⇒ C++ project.
- The "C++ Project" dialog pops up.
  - ➢ In "Project name" field, enter "HelloWorld".
  - ➢ In "Project Types" box, select "Executable" ⇒ "Empty Project".

➢ In "Toolchains" box, choose your compiler, e.g., "Cygwin GCC" or "MinGW GCC" ⇒ Next.

- The "Select Configurations" dialog appears. Select both "Debug" and "Release" ⇒ Finish.

Step 3: Write a Hello-world C++ Program

- In the "Project Explorer" (leftmost panel) ⇒ Right-click on "HelloWorld" (or use the "File" menu) ⇒ New ⇒ Source File.
- The "New Source File" dialog pops up.
  - ➢ In "Source file" field, enter "Hello.cpp".
  - ➢ Click "Finish".
- The source file "Hello.cpp" opens on the editor panel (double-click on "test.cpp" to open if necessary). Enter the following codes:

```
#include <iostream>
using namespace std;

int main() {
cout<< "Hello, world!" <<endl;
    return 0;
}
```

If "Unresolved Inclusion Error"

If error "unresolved inclusion" appears next to #include statement, the "include paths for headers" are not set properly. Select "Project" menu ⇒ Properties ⇒ C/C++ General ⇒ Paths and Symbols ⇒ In "Includes" tab:

For Cygwin GCC:

1. "Add" the following directories to "GNU C", where $CYGWIN_HOME is your Cygwin installed directory:
- $CYGWIN_HOME\lib\gcc\i686-pc-cygwin\4.5.x\include
- $CYGWIN_HOME\lib\gcc\i686-pc-cygwin\4.5.x\include-fixed
- $CYGWIN_HOME\usr\include
- $CYGWIN_HOME\usr\include\w32api

2. "Add" the following directories to "GNU C++", where $CYGWIN_HOME is your Cygwin installed directory:
- $CYGWIN_HOME\lib\gcc\i686-pc-cygwin\4.5.x\include\c++
- $CYGWIN_HOME\lib\gcc\i686-pc-cygwin\4.5.x\include\c++\i686-pc-cygwin
- $CYGWIN_HOME\lib\gcc\i686-pc-cygwin\4.5.x\include\c++\backward
- $CYGWIN_HOME\lib\gcc\i686-pc-cygwin\4.5.x\include
- $CYGWIN_HOME\lib\gcc\i686-pc-cygwin\4.5.x\include-fixed
- $CYGWIN_HOME\usr\include
- $CYGWIN_HOME\usr\include\w32api

For MinGW GCC:

1. "Add" the following directories to "GNU C", where $MINGW_HOME is your MinGW installed directory:
- $MINGW_HOME\lib\gcc\mingw32\4.6.x\include
- $MINGW_HOME\include
- $MINGW_HOME\lib\gcc\mingw32\4.6.x\include-fixed

2. "Add" the following directories to "GNU C++", where $MINGW_HOME is your Cygwin installed directory:
- $MINGW_HOME\lib\gcc\mingw32\4.6.x\include\c++
- $MINGW_HOME\lib\gcc\mingw32\4.6.x\include\c++\mingw32
- $MINGW_HOME\lib\gcc\mingw32\4.6.x\include\c++\backward
- $MINGW_HOME\lib\gcc\mingw32\4.6.x\include
- $MINGW_HOME\include
- $MINGW_HOME\lib\gcc\mingw32\4.6.x\include-fixed

NOTE: To find the header paths, you can do a search on headers such as "stdio.h" (for C) and "iostream" (for C++) under the Cygwin or MinGW installed directory.

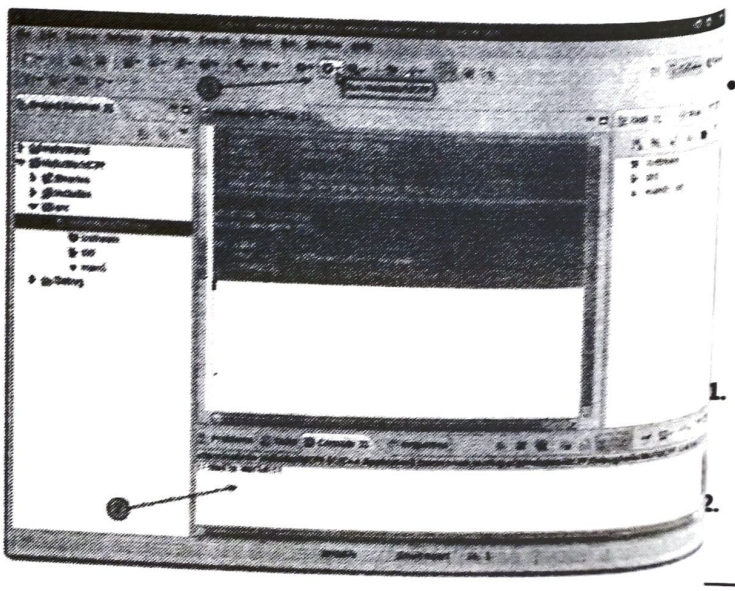
Note: If you encounter "error while loading shared libraries" during link. Install "libmpfr4" in cygwin.

Step 4: Compile/Build

Right-click on the "HelloWorld" (or use the "Project" menu) ⇒ choose "Build Project" to compile and link the program.

Step 5: Run

To run the program, right-click on the "HelloWorld" (or anywhere on the source "test.cpp", or select the "Run" menu) ⇒ Run As ⇒ Local C/C++ Application ⇒ (If ask, choose Cygwin's gdb debugger) ⇒ The output "Hello, world!" appears on the "Console" panel.



1.9 INTRODUCTION TO ALGORITHMS AND FLOWCHARTS

- There are three ways to represent the logical steps for finding the solution to a given problem, i.e., Algorithm, Flowchart and Pseudocode.
- In an algorithm, description of steps for a given problem is provided. A flowchart represents solution to a given problem graphically. Another way to express the solution to a given problem is by means of pseudo code.

1.9.1 Algorithms

- An algorithm is a finite set of precise instructions for performing a computation for solving a problem.
- A step-by-step procedure for solving a particular problem is called as an algorithm.
- Fig. 1.5 shows a simple illustration of how algorithms are used for solving computational problems.
- An algorithm solves only a single problem at a time. However, the same problem can be solved using multiple algorithms.
- The advantage of using multiple algorithms to solve the same problem is purely situational.
- The choice of a particular algorithm solely depends on the type of input values i.e., single or multiple.

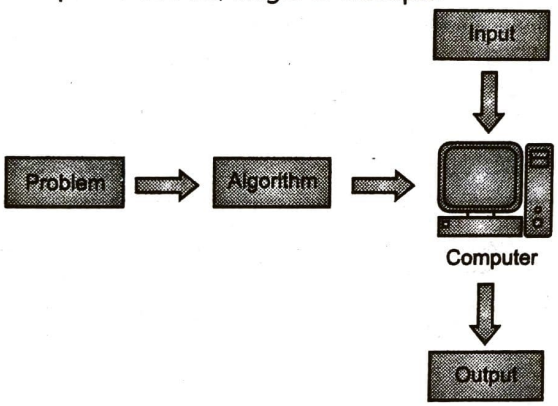


Fig. 1.5: Use of Algorithms for Solving Computational Problems

- Algorithmic strategies can be written using two methods as shown in Fig. 1.6.

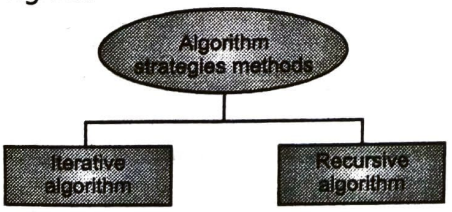


Fig. 1.6: Strategies of algorithm

1. **Iterative Algorithm:** In iterative algorithms, the process is carried out repetitively on the inputs in order to achieve the desired output.
2. **Recursive Algorithm:** This algorithm is an algorithm which calls itself with smaller (or simple) input values, and which obtains the result for the current input by applying the sam operations for the smaller (or simpler) input.

Approaches for Designing an Algorithm:

- Top-down Approach: A top-down design approach starts b identifying the major components of the system or prograr decomposing them into their lower level components an iterating until the desired level of module complexity i achieved. In this we start with the topmost module an incrementally add modules that it calls, (For example: 'C language).
- Bottom-up Approach: A bottom-up design approach start with designing the most basic or primitive components an proceeds to higher level components. Starting from the ver bottom, the operations that provide a layer of abstraction are implemented.

Definition :

- An algorithm can be defined as "a step-by-step procedure that provides solution to a given problem".

OR

- "A set of steps that generates a finite sequence of elementary computational operations leading to the solution of a given problem is called an algorithm".

Characteristics:

Every algorithm must satisfy the following characteristics:

- **Finiteness:** An algorithm must always terminate after a finite number of steps.
- **Definiteness:** Each and every step of the algorithm should be rigorously and unambiguously defined.
- **Input:** The algorithm should take zero or more inputs.
- **Output:** The algorithm should produce one or more outputs,
- **Effectiveness:** A human should be able to calculate the values involved in the procedure of the algorithm using pencil and paper.

Advantages:

- An algorithm gives a language independent layout of the program or problem.
- It allows the programmers to use the most efficient solution of the problem.
- It eases the process of actual development of program code.
- It breaks down the solution of a problem into a series of simplified sequential steps.
- Its simplified way of representing program instructions enables other programmers to easily understand and modify it.

Disadvantages:

- For large algorithms, it becomes difficult to understand the flow of program control.

There are no standard conventions to be followed while developing algorithms.

It may take considerable amount of time to write the algorithm for a given problem.

It lacks the visual representation of programming logic as is prevalent in flowcharts.

1.9.2 Flowcharts

- A flowchart is a visual representation of the sequence of steps for solving a problem.
- A flowchart can be referred as a pictorial representation of an algorithm. The objective of using flowcharts to describe the problem solution is to ease the understanding of programming logic.
- A flowchart is a diagrammatic representation of the algorithm or of the plan of solution of a problem.
- Flowchart indicates the process of solution, the relevant operations and computations, the point of decision and other information which is a part of the solution.

Principles of Flowcharts:

- Pictorial representation of flowchart makes it a convenient method of communication.
- It promotes logical accuracy and is a key to correct programming.
- It takes care that no path is left incomplete without any action being taken.
- It helps to develop program logic and serves as documentation.
- It is an important tool for planning and designing a new system.

Types of Flowcharts:

1. **System Flowchart:** Used by system analyst. This flowchart shows various processes, subsystems, outputs and operations on data in a system.
2. **Program Flowchart:** Used by computer programmers. This flowchart shows program structure, logic flow and operations performed.

Definition

- We can define flowchart as "a symbolic representation of a solution of a particular problem".

OR

- A flowchart is "a pictorial/graphical/symbolic representation of an algorithm".

Flowchart Symbols

- Flowcharts are constructed or designed by using special geometrical symbols. Each symbol represents an activity. The activity could be input/output of data, computation /processing of data, taking a decision, terminating the solution, etc. The symbols are joined by arrows to obtain a

- Following Table 1.1 shows different flowcharting symbols.

Table 1.1: Flowcharting Symbols

Sr. No.	Symbol	Name (Alternates)	Meaning
1.		Process	An operation or action step.
2.		Terminator	A start or stop point in a process or program.
3.		Decision	A question or branch in the process or function.
4.		Predefined process	A formally defined sub-process.
5.		Data (I/O)	Indicates data inputs and outputs (I/O) to and from process.
6.		Document	A document or report.
7.		Multi-document	Same as document but well multiple document.
8.		Preparation	A preparation or set-up process step.
9.		Display	A machine display.
10.		Connector	A jump from one point to another.
11.		Off-page connector	Continuation onto another page.
12.		Merge (Storage)	Merge multi-process into one.
13.		Extract	Extract a measurement.
14.		Stored data	Data storage symbol in flowchart.
15.		Magnetic disk (database)	A database.
16.		Direct access storage	Storage on a hard disk.
17.		Flow Line	Indicates the direction of data flows.

Advantages and Disadvantages

The Benefits (Advantages) of Flowcharts are as Follows:

- Proper Documentation: Flowcharts serve as a good progr documentation, which is needed for various purposes.
- Efficient Coding: The flowcharts act as a guide or bluep during the systems analysis and program developm phase.
- Efficient Program Maintenance: The maintenance operating program becomes easy with the help of flowch

It helps the programmer to put efforts more efficiently on that part.

- Proper Debugging: The flowchart helps in debugging process.
- Communication: Flowcharts are better way of communicating the logic of a system to all concerned.
- Effective Analysis: With the help of flowchart, problem can be analysed in more effective way.

The Disadvantages (Limitations) of Flowchart are as Follows:

- Loss of Technical Details: The essentials of what is done can easily be lost in the technical details of how it is done.
- Complex Logic: Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
- Alterations and Modifications: If alterations are required, the flowchart may require redrawing completely.
- Reproduction: As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
- Time Consuming: Developing and constructing a flowchart is time consuming. Flowchart requires more time to construct.

1.9.3 Comparison (Difference) between Algorithm and Flowchart

Table 1.2

Sr. No.	Algorithm	Flowchart
1.	An algorithm is a finite set of precise instructions for solving a problem.	A flowchart is a graphical/pictorial representation of an algorithm.
2.	It refers to the logic for solving problem.	It is a tool which shows flow of problem solving.
3.	An algorithm is a precise rule (or set of rules) specifying how to solve some problem. Algorithm is stepwise analysis of the work to be done.	A flowchart is a diagram of the sequence of operation in problem solving.
4.	Algorithm gives language indepen-dent layout of the problem.	Flowchart gives logical flow of problem.
5.	Easy to update.	Difficult to update.
6.	Less time required for write an algorithm.	Time consuming to write and draw a flowchart.

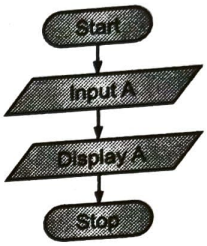
1.10 SIMPLE EXAMPLES ON ALGORITHMS AND FLOWCHARTS

1. Algorithm and Flowchart to Input a Number to the Computer and Display the Same Number on the Screen.

Algorithm:

Step 1 : Start
Step 2 : Enter number A
Step 3 : Display A
Step 4 : Stop

Flowchart:



2. Algorithm and Flowchart to Convert Temperature in Celsius to Fahrenheit.

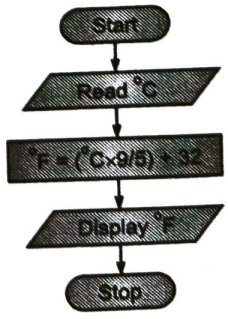
Degree Fahrenheit (°F) and Degree Celsius (°C) are the main two units to measure temperature. It is possible to convert a temperature from Celsius degrees to Fahrenheit using following formula:

$$oF = \left(oC \times \frac{9}{5}\right) + 32$$

Algorithm:

Step 1 : Start
Step 2 : Read oC
Step 3 : Calculate oF = (oC× 9/5) + 32
Step 4 : Display oF
Step 5 : Stop

Flowchart:



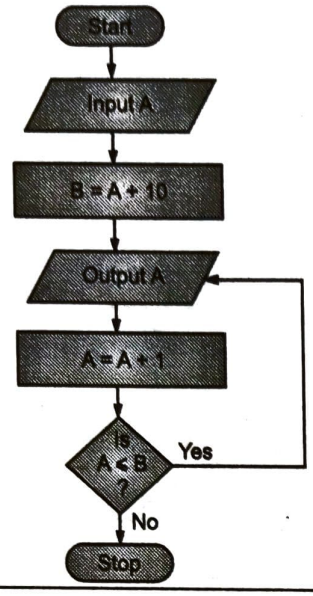
3. Algorithm and Flowchart to Input One Number and Display Next 10 Numbers in Increasing Order.

Algorithm:

Step 1 : Start
Step 2 : Input A
Step 3 : B = A + 10

Step 4 : Output A
Step 5 : A = A + 1
Step 6 : If A < B, Go to Step 4
Step 7 : Stop

Flowchart:



4. Algorithm and Flowchart for Given Year is a Leap Year or Not.

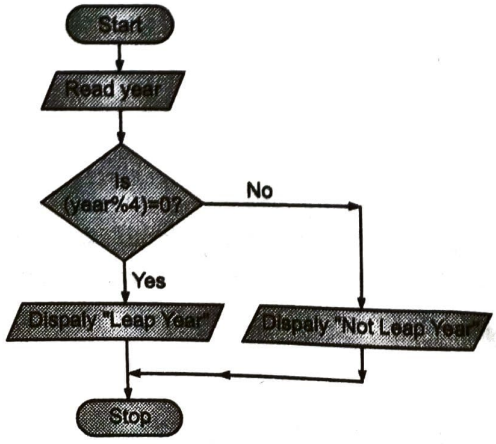
A Year is Called Leap Year if it is Divisible by 400. For Example: 1600, 2000 etc leap year while 1500, 1700 are not leap year.

If year is not divisible by 400 as well as 100 but it is divisible by 4 then that year are also leap year. For example: 2004, 2008, 1012 are leap year.

Algorithm:

Step 1: Start
Step 2: Accept an year value from the user (year)
Step 3: If remainder of year value divided by 4 (year%4) is 0 then go to Step 4 else go to Step 5
Step 4: Display "Leap Year" and go to Step 6
Step 5: Display "Not Leap Year"
Step 6: Stop

Flowchart:

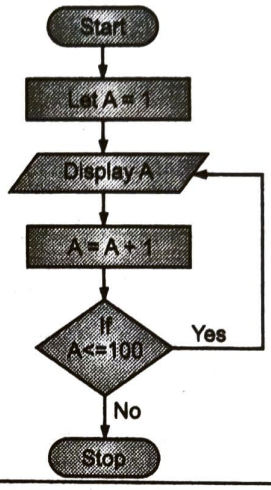


5. Algorithm and Flowchart to Display 1 through 1(Numbers.

Algorithm:

Step 1 : Start
Step 2 : Let A = 1
Step 3 : Display A
Step 4 : A = A + 1
Step 5 : If A < = 100 GotoStep 3
Step 6 : Stop

Flowchart:



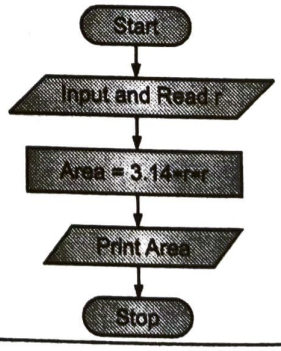
6. Algorithm and Flowchart to Calculate the Area of Circle

Area of circle measures the area enclosed by a circle. formula is A = π *r*r.

Algorithm:

Step 1 : Start
Step 2 : Input and Read r
Step 3 : Area = 3.14 * r * r
Step 4 : Print Area
Step 5 : Stop

Flowchart:



7. Algorithm and Flowchart to Print All Divisors of Inte N.

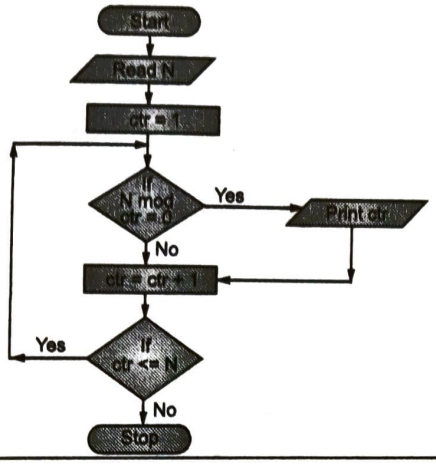
Algorithm:

Step 1 : Start
Step 2 : Read N
Step 3 : Let Ctr = 1
Step 4 : If N Mod Ctr = 0
then N is divisible by Ctr, Print Ctr
Step 5 : Add 1 to Ctr

Step 6' : If Ctr is less than or equal to N then Goto Step 4

Step 7 : Stop

Flowchart:



8. Algorithm and Flowchart for Pass or Fail.

Algorithm:

Step 1 : Start

Step 2 : Input Q1, Q2, Q3, Q4

Step 3 : Grade = (Q1 + Q2 + Q3 + Q4)/4

Step 4 : if (Grade < 50) then

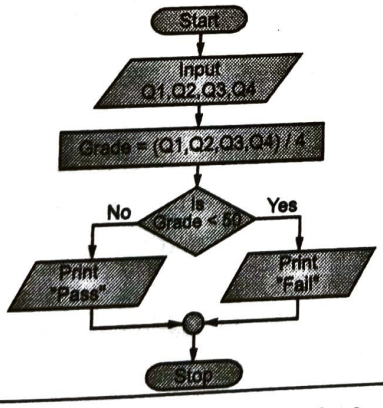
Print "Fail"

else

Print "Pass"

Step 5 : Stop

Flowchart:



9. Algorithm and Flowchart to Convert the Length in Feet to Centimeter.

1 Feet is equal to 30.48 Centimeter.

Algorithm:

Step 1 : Start

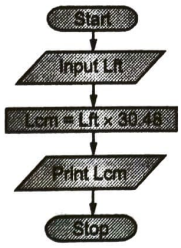
Step 2 : Input Lft

Step 3 : Lcm = Lft× 30.48

Step 4 : Print Lcm

Step 5 : Stop

Flowchart:



10. Algorithm and a Flowchart that will Read the Two Side of a Rectangle and Calculate its Area.

To find the area of a rectangle, multiply the length by th width. The formula is

A = L * W, where A is the area, L is the length, W is th width, and * means multiply.

Algorithm:

Step 1 : Start

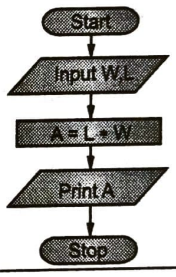
Step 2 : Input W, L

Step 3 : A = L × W

Step 4 : Print A

Step 5 : Stop

Flowchart:



11. Algorithm and Flowchart to Print What do When Drivin to a Traffic Signal.

Algorithm:

Step 1 : Start

Step 2 : Read traffic signal

Step 3 : If signal is GREEN then set action as GO

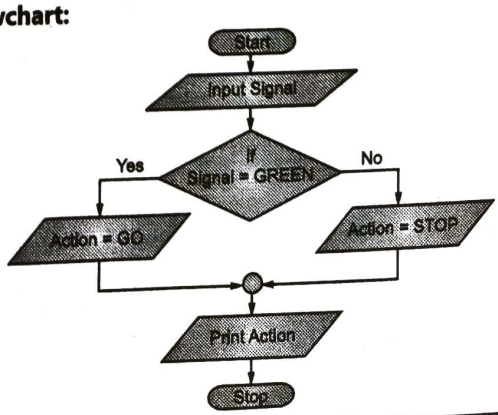
Else

Set Action as STOP

Step 4 : Print Action

Step 5 : Stop

Flowchart:



12. Algorithm and Flowchart to Find the Square of a Number.

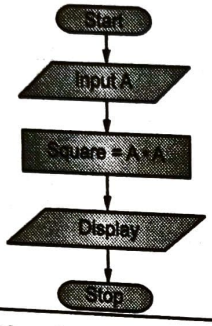
The number we get after multiplying an integer (not a fraction) by itself.

Example: 4 × 4 = 16, so 16 is a square number.

Algorithm:

Step 1 : Start
Step 2 : Input the Number A
Step 3 : Let Square = A * A
Step 4 : Display
Step 5 : Stop

Flowchart:



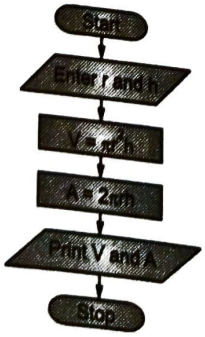
13. Algorithm and Flowchart for Finding the Volume and Surface Area of a Cylinder.

Lateral Surface Area of a Cylinder is directly proportional to the radius and the height of the Cylinder. The formula is A = 2* pi * r * h. There is special formula for finding the volume of a cylinder. The volume is how much space takes up the inside of a cylinder. The formula for calculating volume (V) is π * r * r * h.

Algorithm:

Step 1 : Start
Step 2 : Read radius and height as r and h
Step 3 : Calculate V = π * r * r * h
Step 4 : Calculate A = 2 * π * r * h
Step 5 : Print Volume and Area
Step 6 : Stop

Flowchart:

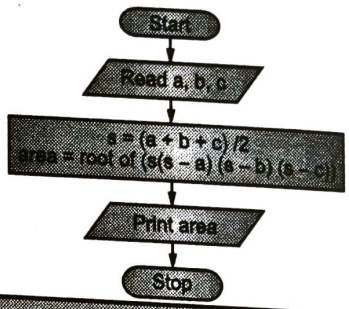


14. Algorithm and Flowchart to Find the Area of a Triang for the Given Three Sides.

Algorithm:

Step 1: Start
Step 2: Declare the floating variables a, b, c, s, area
Step 3: Print the message "enter three sides"
Step 4: Read the values of sides from keyboard
Step 5: Write the logic for s i.e. s = (a + b + c)/2
Step 6: Write the logic for area=root of (s(s – a)(s – b) (s–
Step 7: Print the area.
Step 8: Stop

Flowchart:



EXERCISE

1. Explain the following processes with respect programming
   (i) Editing (ii) Compi
   (iii) Error Checking (iv) Executing the program
2. What do you mean by testing and debugging programs?
3. What is flowchart?
4. Enlist characteristics of algorithm.
5. Define algorithm and flowchart.
6. Enlist advantages and disadvantages of Flowchart.
7. Enlist advantages and disadvantages of an algorithm
8. Differentiate between algorithm and flowchart.
9. Write an algorithm to find odd and even numbers, draw flowchart.
10. Write an algorithm for making tea. Also sketch flow
11. Enlist principles of flowchart.
12. Draw symbols for flowchart.
13. What are the types of flowcharts?
14. Draw flowchart for addition of two numbers.
15. List approaches of algorithms in short.
16. Write algorithm for display the given number is pri not.