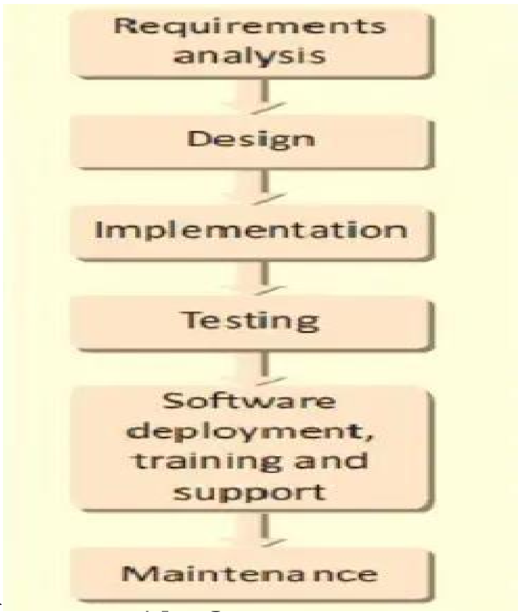
## Design and Implementation of Efficient Programs

- The design and development of correct, efficient and maintainable programs depend on the approach adopted by the programmer to perform various activities that need to be performed during the development process.
- The entire programs and software development process is divided into a number of phases.
- This phase is the Software Development Life cycle.



1. **Requirement analysis:**

- In this phase, user expectations are gathered to know why th
built.
- Then all the gathered requirements are analyzed to frame an objective.
- The last activity in this phase is to document all the requirements to avoid any further doubts.
- The functionality, capability, performance, and availability of hardware and software components are analyzed in this phase.

2. **Design:**

- In this phase, a plan of actions is made before the actual development process could start.
- This plan will be followed throughout the development process.
- The core structure of the software/program is broken down into modules.
- The solution of the program is then specified for each module in the form of algorithms, flowcharts, or pseudo codes.

3. **Implementation:**

- Designed algorithms are converted into program code using any of the high level languages.
- The choice of language depends on the type of program like whether it is a system or an application program.
- Program codes are tested by the programmer to ensure their correctness.

- While constructing the code, the development team checks whether the software is compatible with the available hardware and other software components that were mentioned in the Requirements Specification Document created in the first phase.

**4. Testing:**

- All the modules are tested together to ensure that the overall system works well as a whole product.
- Although individual pieces of codes are already tested by the programmers in the implementation phase, there is always a chance for bugs to creep in the program when the individual modules are integrated to form the overall program structure.
- Software is tested using a large number of varied inputs also known as test data to ensure that the software is working as expected by the users' requirements that were identified in the requirements analysis phase.

**5. Software Deployment, Training and Support:**

- After testing, the software is deployed in the production environment.
- Software Training and Support is a crucial phase which makes the end users familiar with how to use the software.
- Moreover, people are often resistant to change and avoid venturing into an unfamiliar area, so as a part of the deployment phase, it has become very crucial to have training classes for the users of the software.

**6. Maintenance:**

- Maintenance and enhancements are ongoing activities which are done to cope with newly discovered problems or new requirements.
- Such activities may take a long time to complete as the requirement may call for addition of new code that does not fit the original design or an extra piece of code required to fix an unforeseen problem.
- As a general rule, if the cost of the maintenance phase exceeds 25% of the prior- phases cost then it clearly indicates that the overall quality of at least one prior phase is poor.
- In such cases, it is better to re-build the software (or some modules) before maintenance cost is out of control.

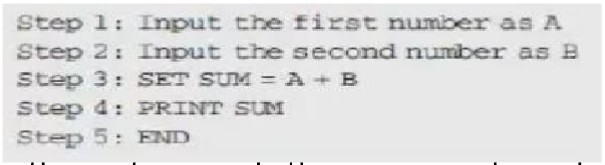
## Algorithms

- An algorithm provides a blueprint to writing a program to solve a particular problem.
- It is considered to be an effective procedure for solving a problem in a finite number of steps.

- A well-defined algorithm always provides an answer, and is guaranteed to terminate.
- Algorithms are mainly used to achieve software re-use.
- A good algorithm must have the following characteristics
    - Be precise
    - Be unambiguous
    - Not even a single instruction must be repeated infinitely
    - After the algorithm gets terminated, the desired result must be obtained

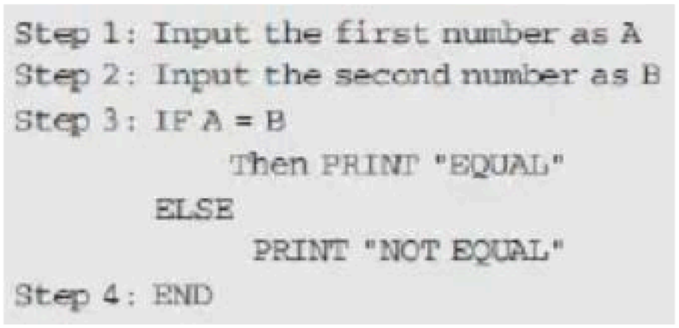
**Control Structures used in Algorithms**

Sequence: Sequence means that each step of the algorithm is executed in the specified order.



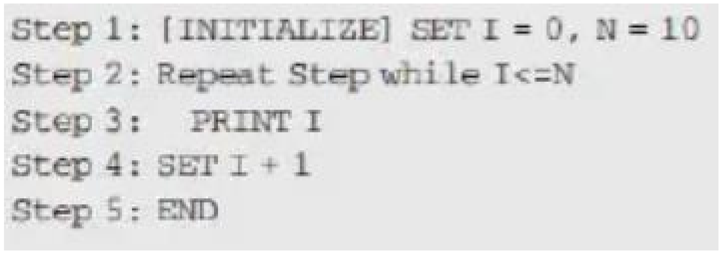
```
Step 1: Input the first number as A
Step 2: Input the second number as B
Step 3: SET SUM = A + B
Step 4: PRINT SUM
Step 5: END
```

Decision: Decision statements are used when the outcome of the process depends on someconditon. For example, if x=y, then print "EQUAL". Hence, the general form of the if construct can be given as if condition then process.



```
Step 1: Input the first number as A
Step 2: Input the second number as B
Step 3: IF A = B
              Then PRINT "EQUAL"
        ELSE
              PRINT "NOT EQUAL"
Step 4: END
```

Repetition: Repetition, which involves executing one or more steps for a number of times, can be implemented using constructs such as while, do-while, and for loops. These loops execute one or more steps until some condition is true.

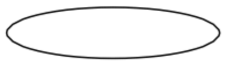
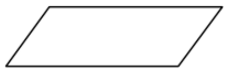
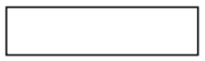
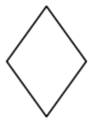
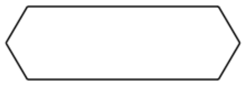
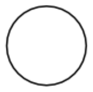
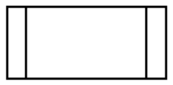


```
Step 1: [INITIALIZE] SET I = 0, N = 10
Step 2: Repeat Step while I<=N
Step 3:    PRINT I
Step 4: SET I + 1
Step 5: END
```

**Note: Please refer to the examples solved in lecture.**

**Flowchart:**

- Flowchart is a graphical or symbolic representation of a process.
- It is basically used to design and document virtually complex processes to help the viewers to visualize the logic of the process, so that they can gain a better understanding of the process and find flaws, bottlenecks, and other less obvious features within it.
- When designing a flowchart, each step in the process is depicted by a different symbol and is associated with a short description. The symbols in the flowchart are linked together with arrows to show the flow of logic in the process.

| **Symbol** | **Name** | **Represents** |
|---|---|---|
|  | Oval | Start and End |
|  | Parallelogram | Input/Output |
|  | Rectangle | Statements or expressions |
|  | Rhombus | Decision making or condition checking |
|  | Hexagon | Looping |
|  | Circle | Connector |
|  | Double ended Rectangle | Functions |

**Note: Please refer to the examples solved in lecture. Pseudocode:**

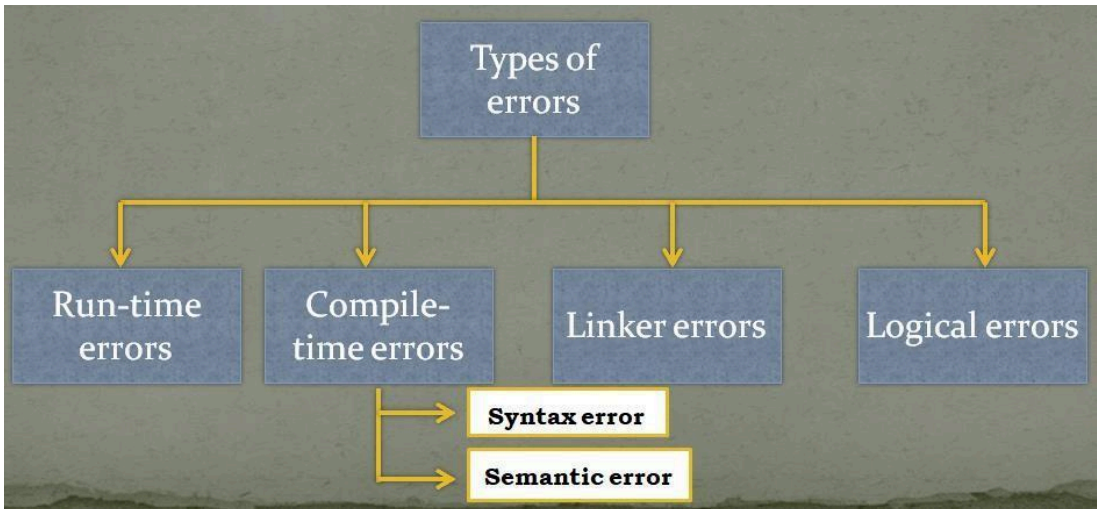
- Pseudocode is a compact and informal high-level description of an algorithm that uses the structural conventions of a programming language.

- It is basically meant for human reading rather than machine reading, so it omits the details that are not essential for humans. Such details include variable declarations, system-specific code, and sub-routines.
- Pseudocodes are an outline of a program that can be easily converted into programming statements.
- They consist of short English phrases that explain specific tasks within a program's algorithm. They should not include keywords in any specific computer language. The sole purpose of pseudocodes is to enhance human understandability of the solution.

**Note: Please refer to the examples solved in lecture.**

**TYPES OF ERRORS**

- While writing programs, very often we get errors in our program.
- These errors if not removed will either give erroneous output or will not let the compiler to compile the program.



1. **Run-time errors:**

- Run-time Errors occur when the program is being run executed.
- Such errors occur when the program performs some illegal operation like

1. Dividing a number by zero
2. Opening a file that already exists

3. Lack of free memory space

4. Finding square or logarithm of negative numbers

- Run-time errors may terminate program execution, so the code must be written in such a way that it handles all sorts of unexpected errors rather terminating it unexpectedly.

- This ability to continue operation of a program despite of run-time errors is called robustness.

**2. Compile time errors:**

- Compile-time Errors occur at the time of compilation of the program.

- Such errors can be further classified as follows:

**Syntax Errors:** Syntax error is generated when rules of C programming language are violated. For example, if we write int a: then a syntax error will occur since the correct statement should be int a;

**Semantic Errors:** Semantic errors are those errors which may comply with rules of the programming language but are not meaningful to the compiler. For example, if we write, a * b = c; it does not seem correct. Rather, if written like c=a*b would have been more meaningful.

**3. Logical errors:**

- Logical Errors are errors in the program code that result in unexpected and undesirable output which is obviously not correct.

- Such errors are not detected by the compiler, and programmers must check their code line by line or use a debugger to locate and rectify the errors.

- Logical errors occur due to incorrect statements. For example, if you meant to perform c= a + b; and by mistake you typed c = a * b; then though this statement is syntactically correct it is logically wrong.

**4. Linker errors:**

- Linker Errors occur when the linker is not able to find the function definition for a given prototype.

- For example, if you write clrscr(); but do not include conio.h then a linker error will be shown.

**Testing Approaches**

- Testing is an activity that is performed to verify correct behavior of a program.
- It is specifically carried out with an intent to find errors.
- Unit testing is applied only on a single unit or module to ensure whether it exhibits the expected behavior.
- Integration Tests are a logical extension of unit tests. In this test, two units that have already been tested are combined into a component and the interface between them is tested. This process is repeated until all the modules are tested together. The main focus of integration testing is to identify errors that occur when the units are combined.
- System testing checks the entire system. For example, if our program code consists of three modules then each of the module is tested individually using unit tests and then system test is applied to test this entire system as one system.

**Debugging Approaches**

- Debugging is an activity that includes execution testing and code correction.
- It is done to locate errors in the program code.
- Once located, errors are then isolated and fixed to produce an error-free code.
- Different approaches applied for debugging a code includes:

**Brute-Force Method:** In this technique, a printout of CPU registers and relevant memory locations is taken, studied, and documented.

It is the least efficient way of debugging a program and is generally done when all the other methods fail.

**Backtracking Method** works by locating the first symptom of error and then trace backward across the entire source code until the real cause of error is detected.

However, the main drawback of this approach is that with increase in number of source code lines, the possible backward paths become too large to manage.

Because Elimination lists all possible causes of an error is developed. Then relevant

tests are carried out to eliminate each of them.

If some tests indicate that a particular cause maybe responsible for an error then the data are refined to isolate the error.

**Introduction to C**

- C was developed in the early 1970s by Dennis Ritchie at Bell Laboratories.
- C was initially developed for writing system software
- Today, C has become a popular language and various software programs are written using this language.
- Many other commonly used programming languages such as C++ and Java are also based on C.

**Characteristics of C**

- C is a high level programming language which enables the programmer to concentrate on the problem and not to worry about the machine code
- Small size: C has only 32 keywords. This makes it relatively easy to learn
- C makes extensive use of function calls
- C is well suited for structured programming. In this programming approach, where it allows the user to think a problem in terms of modules where collection of all these modules make up a complete program.
- Unlike PASCAL it supports loose typing (as a character can be treated as an integer and vice versa)
- Stable language.
- Quick language: Since C programs make use of operators and data types, they are fast and efficient.
- Facilitates low level (bitwise) programming
- Supports pointers to refer computer memory, array, structures and functions.
- C is a core language as many other programming languages are based on C
- C is a portable language which means C program written on one computer can run on another computer with little or no modification.

• C is an extensible language as it enables the user to add his own functions to the C library.

**Uses of C**

• C language is primarily used for system programming. The portability, efficiency, the ability to access specific hardware addresses and low runtime demand on system resources makes it a good choice for implementing operating systems and embedded system applications.

• C has been so widely accepted by professionals that compilers, libraries, and interpreters of other programming languages are often implemented in C.

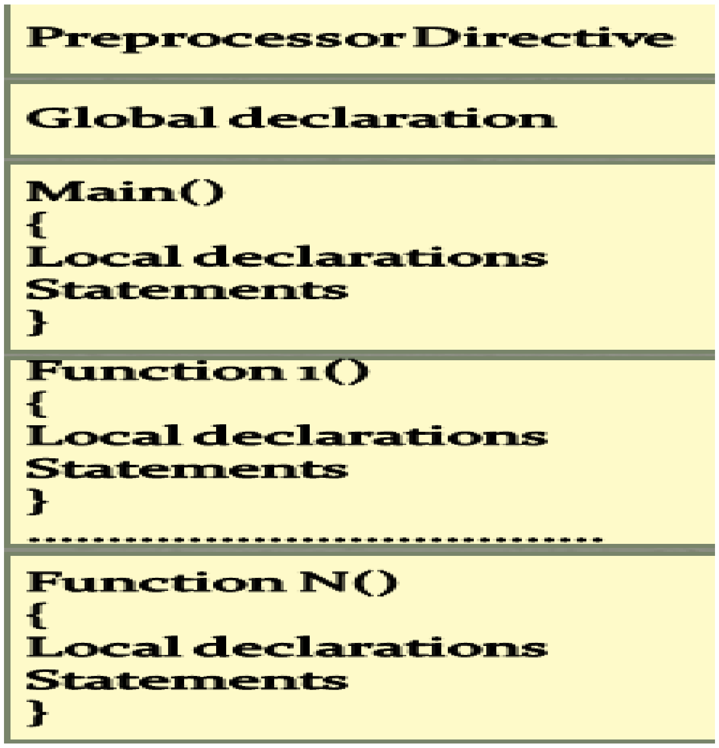
• For portability and convenience reasons, C is sometimes used as an intermediate language by implementations of other languages. Example of compilers which use C this way are BitC, Gambit, the Glasgow Haskell Compiler, Squeak, and Vala.

• C is widely used to implement end-user applications

**Structure of a C program**

- A C program contains one or more functions
- The statements in a C program are written in a logical sequence to perform a specific task.
- Execution of a C program begins at the main() function
- You can choose any name for the functions.
- Every program must contain one function that has its name as main().

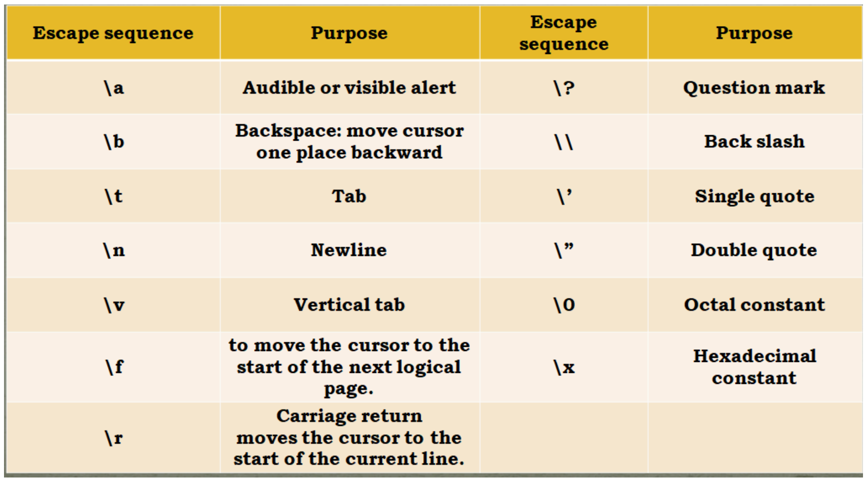
Nisha



```
Preprocessor Directive

Global declaration

Main()
{
Local declarations
Statements
}

Function 1()
{
Local declarations
Statements
}
....................................

Function N()
{
Local declarations
Statements
}
```

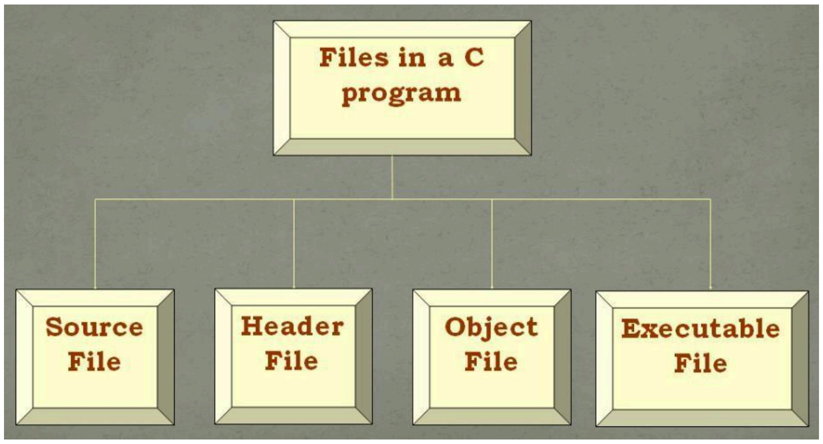
Where, #include refers to the preprocessor statement used for the standard input/output functions and it starts with the hash (#) symbol.

main( ):- Refers to the function from where the execution of the program begins.



| Escape sequence | Purpose | Escape sequence | Purpose |
|---|---|---|---|
| \a | Audible or visible alert | \? | Question mark |
| \b | Backspace: move cursor one place backward | \\ | Back slash |
| \t | Tab | \’ | Single quote |
| \n | Newline | \” | Double quote |
| \v | Vertical tab | \0 | Octal constant |
| \f | to move the cursor to the start of the next logical page. | \x | Hexadecimal constant |
| \r | Carriage return moves the cursor to the start of the current line. | | |

## Files used in C program



## Source code file

- A source code is a list of commands that has to be assembled or compiled into an executable computer program.
- The source code file contains the source code of the program. The file extension of any C source code file is “.c”.
- This file contains C source code that defines the *main* function and maybe other functions.

- The main() is the starting point of execution when you successfully compile and run the program.
- A C program in general may include even other source code files (with the file extension .c).

**Header file**

- When working with large projects, it is often desirable to make sub-routines and store them in a different file known as header file. The advantage of header files can be realized when

a) The programmer wants to use the same subroutines in different programs.

b) The programmer wants to change, or add, subroutines, and have those changes be reflected in all other programs.

- Conventionally, header files names ends with a ".h" extension and its name can use only letters, digits, dashes, and underscores.
- While some standard header files are available in C, but the programmer may also create his own user defined header files

**Standard Header file**

Examples of standard header files are:

1. string.h: for string handling functions.
2. stdlib.h: for some miscellaneous functions.
3. stdio.h: for standardized input and output functions.
4. math.h: for mathematical functions
5. alloc.h: for dynamic memory allocation.
6. conio.h: for clearing the screen

**Object Files:**

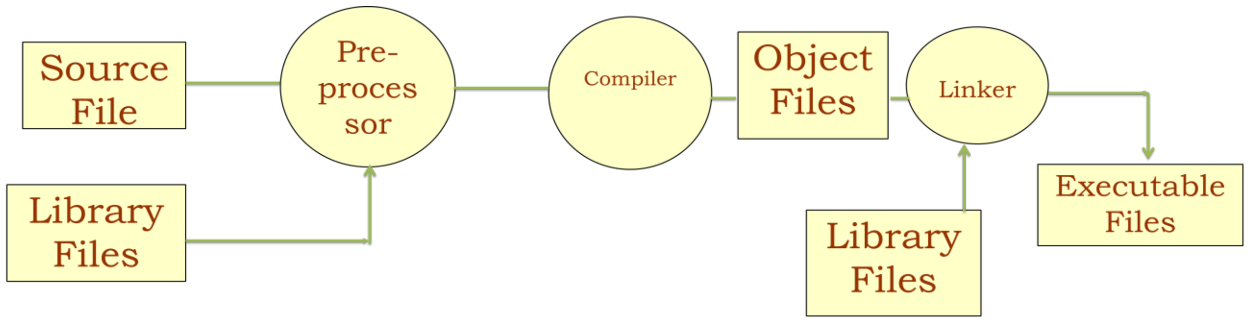
- Object files are generated by the compiler as a result of processing the source code file.
- Object files contain compact binary code of the function definitions.

- Linker uses this object file to produce an executable file (.exe file) by combining the of object files together.
- Object files have a “.o” extension, although some operating systems including Windows and MS-DOS have a “.obj” extension for the object file.

**Binary Executable File**

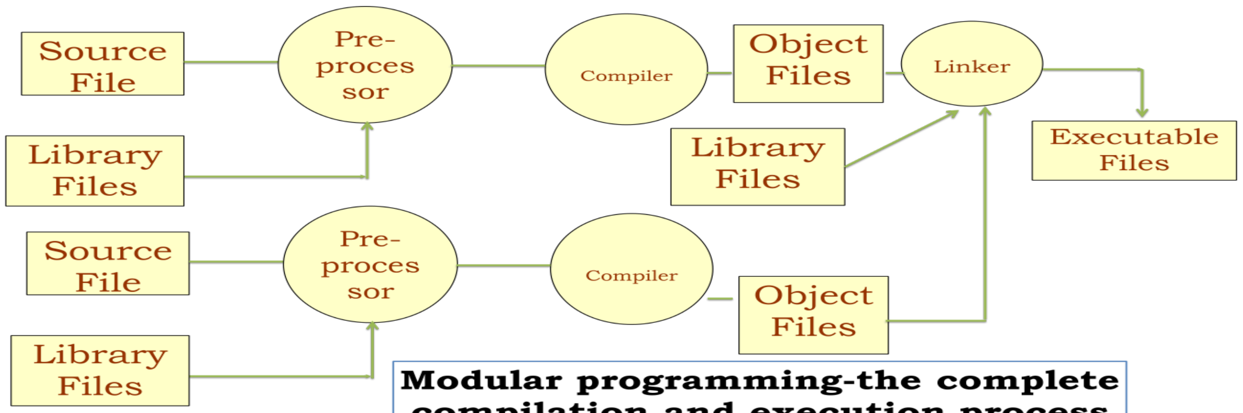
- The binary executable file is generated by the linker.
- The linker links the various object files to produce a binary file that can be directly executed.
- On Windows operating system, the executable files have “.exe” extension.

**COMPILING AND EXECUTING C PROGRAMS**



**Preprocessing before compilation**

- The compiler translates the source code into an object code.
- The object code contains the machine instructions for the CPU, and calls to the operating system API(Application Programming Interface).
- However, even the object file is not a executable file.
- Therefore, in next step, the object file is processed with another special program called a linker.
- While there is a different compiler for every individual language, the same linker is used for object files regardless of the original language in which the new program was written.
- The output of the linker is an executable or runnable file.



**Modular programming-the complete compilation and execution process**

**Using Comments**

- It is a good programming practice to place some comments in the code to help the reader understand the code clearly.

- Comments are just a way of explaining what a program does. It is merely an internal program documentation.

- The compiler ignores the comments when forming the object file. This means that the comments are non-executable statements.

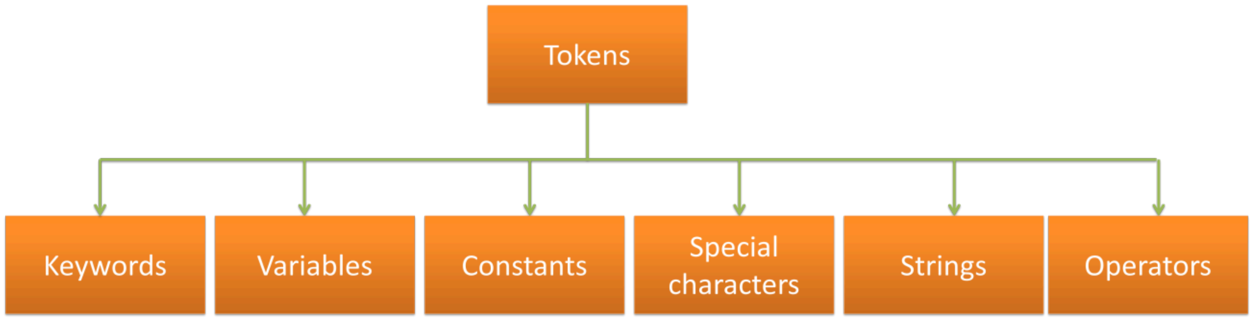
C supports two types of commenting.

- // is used to comment a single statement. This is known as a line comment. A line comment can be placed anywhere on the line and it does not require to be specifically ended as the end of the line automatically ends the line.

- /* is used to comment multiple statements. A /* is ended with */ and all statements that lie within these characters are commented.

**C tokens**

- Tokens are the basic building blocks in C language.
- It is the smallest individual unit in C program.
- A program is constructed using a combination of these tokens. 6main types of tokens in C are:



**Character set in C**

- In C, character means any letter from English alphabet, a digit or a special symbol used to represent information.

- The character set of C can therefore be given as:

1. English alphabet: Include both lower case (a-z) as well upper case(A-Z) letters.

2. Digits: Include numerical digits from 0 to 9.

3. Special characters: Include symbols such as ~, @, %, ^, &, *, {, }, <, >, =, _, +, -, $, /, (, ), \, ;, : , [, ], ‘, “, ?, ., !, |.

4. White space characters: These characters are used to print a blank space on the screen like \b, \t, \v, \r, \f, \n.

5. Escape sequences.

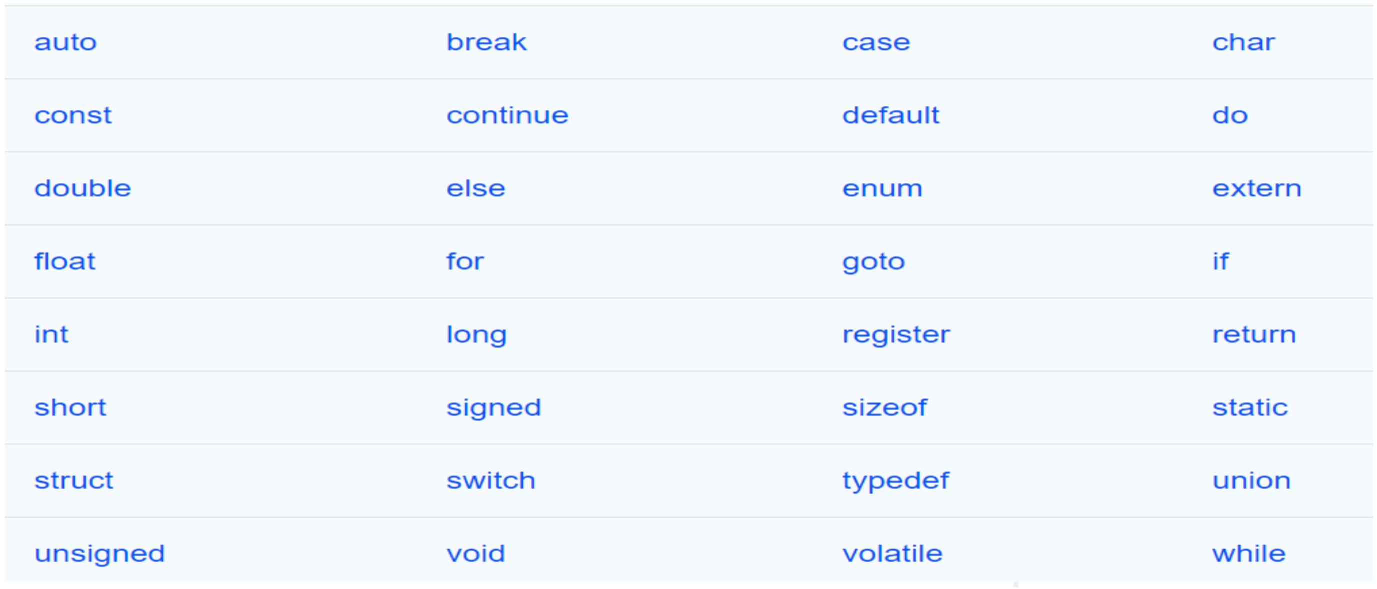
**Keywords**

- C has a set of 32 reserved words often known as keywords.

- All keywords are basically a sequence of characters that have a fixed meaning.

- By convention all keywords must be written in lowercase (small) letters.

- The keywords are for, while, do-while, auto break, case, char, continue, do, double, else, enum, extern, float, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile



| | | | |
|---|---|---|---|
| auto | break | case | char |
| const | continue | default | do |
| double | else | enum | extern |
| float | for | goto | if |
| int | long | register | return |
| short | signed | sizeof | static |
| struct | switch | typedef | union |
| unsigned | void | volatile | while |

**Keywords in C**

**Identifiers**

- Identifiers are names given to program elements such as variables, arrays and functions.

Rules for forming identifier name

1. It cannot include any special characters or punctuation marks (like #, $, ^, ?, ., etc) except the underscore"_".
2. There cannot be two successive underscores
3. Keywords cannot be used as identifiers
4. The names are case sensitive. So, example, “FIRST” is different from “first” and “First”.
5. It must begin with an alphabet or an underscore.

It can be of any reasonable length. Though it should not contain more than 31 characters.

Example: roll_number, marks, name, emp_number, basic_pay, HRA, DA, dept_code, a , sum, temp.

## Basic data types in C

| Data type | DATA TYPE | SIZE IN BYTES | RANGE | Use |
|---|---|---|---|---|
| Character | char | 1 | -128 to 127 | To store characters |
| Integer | int | 2 | -32768 to 32767 | To store integer numbers |
| Floating point | float | 4 | 3.4E-38 to 3.4E+38 | To store floating point numbers |
| Double | double | 8 | 1.7E-308 to 1.7E+308 | To store big floating point numbers |
| Valueless | void | 10 | 3.4E-4932 to 1.1E+4932 | - |

| DATA TYPE | SIZE IN BYTES | RANGE |
|---|---|---|
| char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| signed char | 1 | -128 to 127 |

Nish

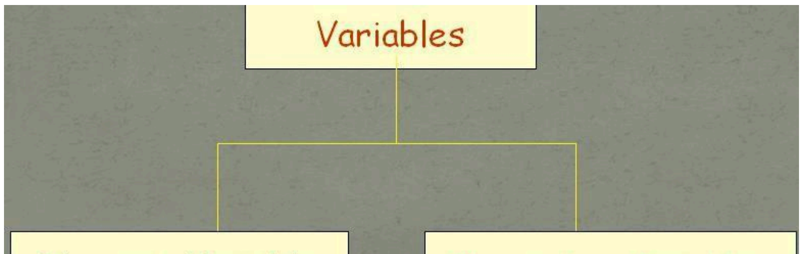
| | | |
|---|---|---|
| **int** | **2** | **-32768 to 32767** |
| **unsigned int** | **2** | **0 to 65535** |
| **signed short int** | **2** | **-32768 to 32767** |
| **signed int** | **2** | **-32768 to 32767** |

| | | |
|---|---|---|
| **short int** | **2** | **-32768 to 32767** |
| **unsigned short int** | **2** | **0 to 65535** |
| **long int** | **4** | **-2147483648 to 2147483647** |
| **unsigned long int** | **4** | **0 to 4294967295** |
| **signed long int** | **4** | **-2147483648 to 2147483647** |
| **float** | **4** | **3.4E-38 to 3.4E+38** |
| **double** | **8** | **1.7E-308 to 1.7E+308** |
| **long double** | **10** | **3.4E-4932 to 1.1E+4932** |

**Variables**

• A variable is defined as a meaningful name given to the data storage location in computer memory.

• When using a variable, we actually refer to address of the memory where the data is stored. C language supports two basic kinds of variables.



- Numeric variables can be used to store either integer values or floating point values.

- While an integer value is a whole numbers without a fraction part or decimal point, a floating point number, can have a decimal point in them.

- Numeric values may also be associated with modifiers like short, long, signed and unsigned.
- By default, C automatically a numeric variable signed..
- Character variables can include any letter from the alphabet or from the ASCII chart and numbers 0 – 9 that are put between single quotes.

**Declaring Variables**

- To declare a variable specify data type of the variable followed by its name.
- Variable names should always be meaningful and must reflect the purpose of their usage in the program.
- Variable declaration always ends with a semicolon. Example, int

emp_num;

float salary; char

grade;

double balance_amount;

unsigned short int acc_no;

**Initializing Variables**

- Assigning a value to a variable is called variable initialization.
- The syntax is variable_name=value; Initialization can be done in two ways:-

1. We can assign an initial value to a variable within the declaration part itself called static initialization.

Ex:- int a=10;, float x=100.0000;

2. We can assign an initial value to a variable during the program execution called dynamic initialization.

```
Ex:-    printf(Enter a value to a");
        scanf("%d", &a);
```

**Constants**

- Constants are identifiers whose value does not change.
- Constants are used to define fixed values like PI or the charge on an electron so that their value does not get changed in the program even by mistake.
- There are mainly four types of constants in C:

1.Integer constants

2.Floating-point constants

3.Character constants

4.String constants

To declare a constant, precede the normal variable declaration with const keyword and assign it a value. For example,

- const float pi = 3.14;

Another way to designate a constant is to use the pre-processor command define.

- #define PI 3.14159
- **The integer constants:** represents a whole number. Ex:-

10, -200, -3456.

2. **The floating-point constants:-**represents a number that has a decimal point. Ex:- -

250.01, 67.73234.

3. **The character constants:** represents a single character enclosed within a pair of single quotes.

Ex.:- ‘s’, ‘%’, ‘7’.

4. **The string constants:-**represents a sequence of characters enclosed within a pair of double quotes.

Ex:- “Hello”, “z”.

## Declaring Constants

Rules that need to be applied to a #define statement which defines a constant.

**Rule 1:** Constant names are usually written in capital letters to visually distinguish them from other variable names which are normally written in lower case characters. Note that this is just a convention and not a rule.

**Rule 2:** No blank spaces are permitted in between the # symbol and define keyword **Rule 3:** Blank

space must be used between #define and constant name and between constant name and constant

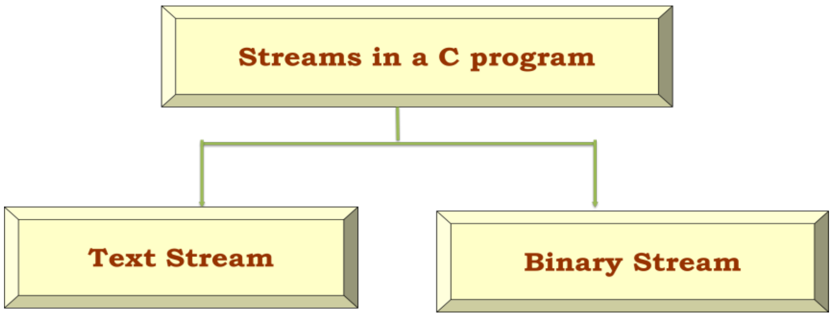
value

**Rule 4:** #define is a pre-processor compiler directive and not a statement. Therefore, it does not end with a semi-colon.

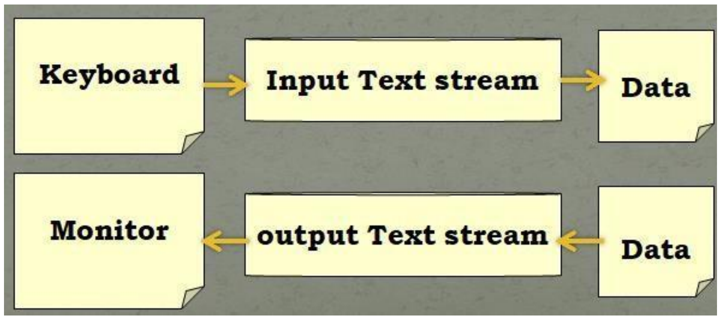
## Input/Output statements in C

**Streams:**

- A stream acts in two ways. It is the source of data as well as the destination of data.
- C programs input data and output data from a stream.
- Streams are associated with a physical device such as the monitor or with a file stored on the secondary memory.



- In a text stream, sequence of characters is divided into lines with each line being terminated with a new-line character (\n). On the other hand, a binary stream contains data values using their memory representation.
- Although, we can do input/output from the keyboard/monitor or from any file but here we will assume that the source of data is the keyboard and destination of the data is the monitor.



**Formatting Input/output:**

- C supports two formatting functions.
- printf is used to convert data stored in the program into a text stream for output to monitor.
- scanf is used to convert the text stream coming from the keyboard to data values and store them in program variables.

**printf():**

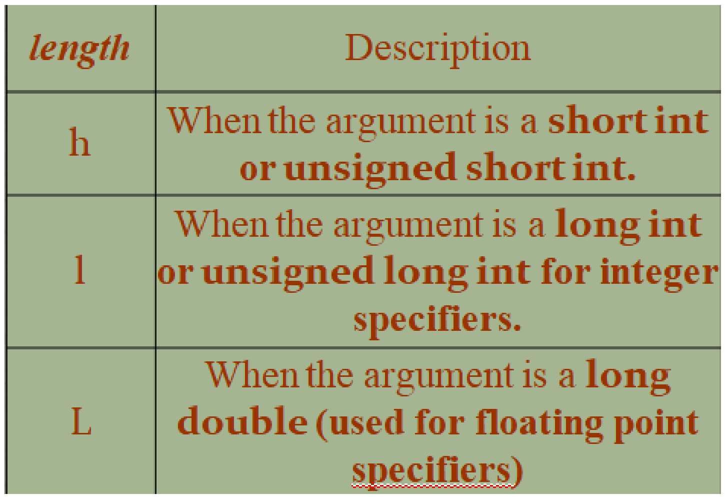
- The printf() function is used to display information required to the user and also prints the values and variables.
- The syntax can be given as printf("Control string", variable list);
- The prototype of the control string is given by

**%[flags][width][.precision][modifier]type**

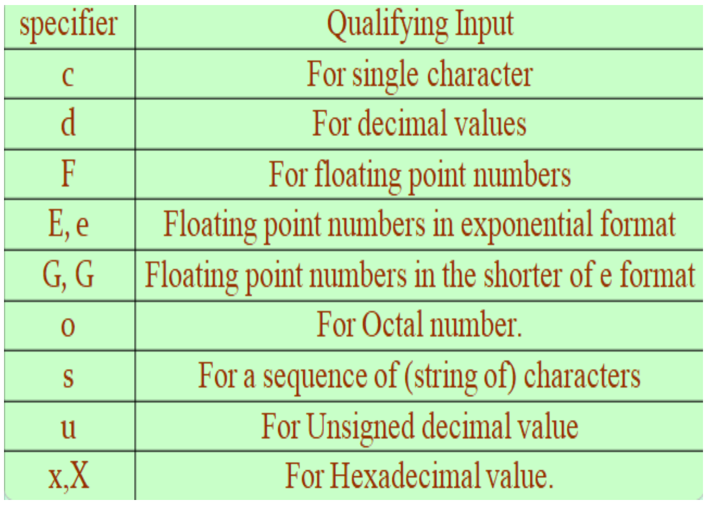
**Flags** are an optional argument which specifies output justification such as numerical sign, trailing zeros or octal, decimal, or hexadecimal prefixes.

| *flag* | description |
|---|---|
| - | Left-justify within the data given field width |
| + | Displays the data with its numeric sign (either + or -) |
| # | Used to provide additional specifiers like **o, x, X, o, ox or oX for octal and hexa decimal values respectively for values different than zero.** |
| o | The number is left-padded with zeroes (**o**) **instead of spaces** |

**Width** is an optional argument which specifies minimum number of positions in the output.
**Precision** is an optional argument which specifies the maximum number of characters to print.



| *length* | Description |
|---|---|
| h | When the argument is a **short int or unsigned short int.** |
| l | When the argument is a **long int or unsigned long int for integer specifiers.** |
| L | When the argument is a **long double (used for floating point specifiers)** |



| specifier | Qualifying Input |
|---|---|
| c | For single character |
| d | For decimal values |
| F | For floating point numbers |
| E, e | Floating point numbers in exponential format |
| G, G | Floating point numbers in the shorter of e format |
| o | For Octal number. |
| s | For a sequence of (string of) characters |
| u | For Unsigned decimal value |
| x,X | For Hexadecimal value. |

**Example:**
%6.3f means print floating point value of max 6 digits where 3 digits are allotted for the digits after the decimal point.

printf("\n Result: %6.3f", 123.53435); output:

Result: 123.534

**Some examples of printf:**

- printf("\n Result: %d%c%f",12,'a',2.3);

  **Output:** Result: 12a2.300000

- printf("\n Result: %d%c%3.1f",12,'a',2.3);

  **Output:** Result: 12a2.3

- printf("\n Result: %d %c %f",12,'a',2.3);

  **Output:** Result: 12 a 2.300000

- printf("\n Result: %5d %x %#x",232,232,232);

  **Output:** Result:        232 e8 0xe8

- printf(“\n Result: %5d %X %#X”,232,232,232);

**Output:** Result:   232 E8 0XE8

- printf("the number is \n%d\n%2d\n%3d",1,1,1);

**Output:**

```
the number is 1
 1
  1
```

- printf("the number is %09.2f",123.456);

**Output:** the number is 000123.46

- printf(“\n%7.4f\n%7.2f\n%-7.2f\n%f\n%10.2e\n%11.4e\n%- 10.2e\n%e”,98.7654, 98.7654, 98.7654, 98.7654, 98.7654, 98.7654, 98.7654, 98.7654);

**Output:**

```
98.7654
  98.77
98.77
98.765400
  9.88e+01
 9.8765e+01
9.88e+01
9.876540e+01
```

**scanf():**

- The scanf() function is used to read formatted data from the keyboard.

- The syntax is scanf(“control string”,arg1,arg2,arg3);
- The prototype of control string is

**%[*][width][modifier]type**

**Rules to use scanf():**

- Rule 1: Rule 1: The scan function works until:
  (a) the maximum number of characters has been processed
  (b) a white space character is encountered, or
  (c) an error is detected.
- Rule 2: Every variable that has to be processed must have a conversion specification associated with it. Therefore, the following scan statement will generate an error as num3 has no conversion specification associated with it. scanf("%d %d", &num1, &num2, &num3) ;
- Rule 3: There must be a variable address for each conversion specification. Therefore, the following scanf statement will generate an error as no variable address is given for the third conversion specification.
  scanf(*'%d %d %d", &numa, &num2) ;
  Remember that the ampersand operator (&) before each variable name specifies the address of that variable name.
- Rule 4: An error will be generated if the format string is ended with a white space character.
- Rule 5: The data entered by the user must match the character specified in the control string (except white space or a conversion specification), otherwise an error will be generated and scan will stop its processing.

For example, consider the following scan statement.
scanf(*%d / %d", &num1, &num2) ; Here, the slash in the control string is neither a white space character nor a part of conversion specification, so the users must enter data of the form 21/46.
Rule 6: Input data values must be separated by spaces.
Rule 7: Any unread data value will be considered as a part of the data input in the next call to scanf.
Rule 8: When the field width specifier is used, it should be large enough to contain the input data size.

*****End*****