Computational Physics

By
Dr. Mahesh M. KAMBLE
PDEA’S Anantrao Pawar College, Pirangut, Mulshi, Pune 412115
Affiliated to
Savitribai Phule Pune University, Pune
Topics Covered

1. Concepts of programming
2. C Programming
3. Arrays and Pointers in C
4. User Defined Function in C
5. Graphics in C
Computer: A device that operates on information or data

Characteristics: Speed, Diligence, Versatility and Power of remembering (storage)

Hardware: Physical device that make up the computer system

1. input devices (keyboard, Mouse, joystick, bar code reader etc.
2. output devices (visual display unit, Printer),
3. Central Processing Unit
4. Memory devices

Software: collections of programs and Applications that manage and work with hardware

System Hardware (Operating system)
Application software (Word processor graphic software)

Languages

Logic

Dr. Mahesh M. Kamble
Algorithms: it is finite sequence of instructions, which can be carried out to solve particular problem in order to obtain the desired result.

Characteristics:
- Definiteness: precise, clear and unambiguous.
- Properly organized.
- Finiteness: must terminate after finite number of steps
- Input: Zero or more inputs
- Output: One or more outputs
- Effectiveness: guaranteed to give correct answer
  Independent on programming language

Advantages:
- general purpose tools and it is language independent
- It is easy to understand program language
- It is written in English
- Identification of error is easier

Limitations:
- Time consuming process
- It is difficult to show branching and repetitive task in algorithm
- For big task, it is lengthy and complicated

Dr. Mahesh M. Kamble
Flowchart: It is the pictorial representation of an algorithm that uses boxes of different shapes to denote different instructions.

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

- Better communications
- Effective coding
- Effective analysis
- Systematic debugging
- Effective Synthesis
- Proper Program documentation
- Language independent

Limitations of Flow chart:

- Time consuming
- Any change or modifications in flowchart is difficult
- Very often, it is very lengthy
**Programming Languages:** it is a set of notations for writing programs, which are specifications of flowchart or algorithm.

**Low Level Language:** Machine level programming language, Assembly language

**Assembly Language:** is a set of instructions executed directly by a computer’s central processing unit.

**High-Level Language:** is a programming language with strong abstraction from the details of the computer such as registers, memory addresses and call stacks.

**Middle level programming languages:** closely related to machine level as well as user friendly.
Getting started with C

Steps in learning English Language

1. Alphabets
2. Words
3. Sentences
4. Paragraph

Steps in learning ‘C’ Language

1. Alphabets
2. Digits
3. Special Symbols
4. Constants
5. Variable
6. Keywords
7. Instructions
8. Program

Dr. Mahesh M. Kamble
THANK YOU!
Chapter 2...

C Programming

Dr. Mahesh M. KAMBLE
PDEA’S Anantrao Pawar College, Pirangut, Mulshi, Pune
INTRODUCTION

History of C

C is programming language developed at AT & T’s Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie.

Importance of C:

- C is general Purpose, structured programming language
- C is reliable, simple and easy to use
- The set of instructions in C are general words of English such as, if, else, while, do, for etc.
- It bridges the gap between machine language (0’s &1’s) and higher level language.
- C language has very good portability
- Programs written in C are very concise
2.1 Structure of a C Program

- Every C Program consists of one or more functions, one of the function must be the ‘main()’ function.

Each function must contain:

1. A function heading, which consists of function name followed by optional list of arguments, enclosed in parenthesis.
2. A list of argument declaration.
3. A compound statement, which forms remaining part of function.

Each compound statement is enclosed in pair of braces i.e. {   }.
Structure of a C Program

Comments: /* and */

#include <stdio.h>

#include – preprocessor

stdio.h - standard input output header file

Sample of C Program:

1. /* My first program in C */

2. #include<stdio.h>

3. Main()

4. {

5. printf("Hello!, This is my first program \n")

6. }
Sample of C Program:

/* C program to calculate area of circle*/ = /*Comment*/
#include<stdio.h> /*library file access*/
Main() /*function heading*/
{
    float radius, area; /*Variable declaration*/
    printf("enter the value of radius = "); /*output statement (prompt)*/
    scanf("%f ", & radius); /*Input statement */
    Area=3.14*radius*radius; /*Assignment statement*/
    printf("area= %f ", area); /*output statement */
}

Output:
enter the value of radius =3
Area = 28.274309
2.2 character Set

Used to form words, numbers and expressions

1. Letters

2. Digits

3. Special Characters

4. White space

<table>
<thead>
<tr>
<th>Letters</th>
<th>Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower case a------z</td>
<td>All decimal digits 0---9</td>
</tr>
<tr>
<td>Upper case A------Z</td>
<td></td>
</tr>
</tbody>
</table>
2.2 character Set

3. Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Name</th>
<th>Character</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>Comma</td>
<td>&amp;</td>
<td>Ampersand</td>
</tr>
<tr>
<td>.</td>
<td>Period</td>
<td>^</td>
<td>Caret</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon</td>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td>:</td>
<td>Colon</td>
<td>-</td>
<td>Minus</td>
</tr>
<tr>
<td>?</td>
<td>Question mark</td>
<td>+</td>
<td>Plus</td>
</tr>
<tr>
<td>'</td>
<td>Single quote</td>
<td>=</td>
<td>Equal</td>
</tr>
<tr>
<td>“</td>
<td>Double quote</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>!</td>
<td>Exclamation</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td></td>
<td>Vertical bar</td>
<td>(</td>
<td>Left parenthesis</td>
</tr>
<tr>
<td>/</td>
<td>Slash</td>
<td>)</td>
<td>Right parenthesis</td>
</tr>
<tr>
<td>\</td>
<td>Back slash</td>
<td>[</td>
<td>Left square bracket</td>
</tr>
<tr>
<td>~</td>
<td>Tilde</td>
<td>]</td>
<td>Right square bracket</td>
</tr>
<tr>
<td>_</td>
<td>Underscore</td>
<td>{</td>
<td>Left curly bracket</td>
</tr>
<tr>
<td>$</td>
<td>Dollar</td>
<td>}</td>
<td>Right curly bracket</td>
</tr>
<tr>
<td>%</td>
<td>Percentage</td>
<td>#</td>
<td>Hash-Number sign</td>
</tr>
</tbody>
</table>

4. White Space, Blank Space, Horizontal Tab, Carriage return, new line, form feed
C Tokens

The smallest individual unit in c program are known as C token

<table>
<thead>
<tr>
<th>Token</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Keyword</td>
<td>int, char</td>
</tr>
<tr>
<td>2. Identifier</td>
<td>main, amount</td>
</tr>
<tr>
<td>3. Constants</td>
<td>3.14, 9.80</td>
</tr>
<tr>
<td>4. String</td>
<td>“Hellow”, “C”</td>
</tr>
<tr>
<td>5. Special symbol</td>
<td>[ ], { }</td>
</tr>
<tr>
<td>6. Operators</td>
<td>+, -, *, /</td>
</tr>
</tbody>
</table>
2.3 Keywords and identifiers

Keywords: keywords are reserved words which cannot be used in program as variable names. All keywords have standard and predefined meaning.

<table>
<thead>
<tr>
<th>Keywords in C Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
</tr>
<tr>
<td>break</td>
</tr>
<tr>
<td>case</td>
</tr>
<tr>
<td>char</td>
</tr>
<tr>
<td>continue</td>
</tr>
<tr>
<td>do</td>
</tr>
<tr>
<td>default</td>
</tr>
<tr>
<td>const</td>
</tr>
</tbody>
</table>
Identifiers: Identifiers are the names given to various program elements such as variables, functions and arrays. These are user defined names. It consists of a sequence of letters and digits with letters as the first character. Lower case letters are commonly used. Underscore character is also permitted.

E.g. x, y1 x12sum_1, tax_free, PRIME

Invalid Identifiers 5th, “W”, roll-no, serial number
2.4 Data Types

All C complier support five fundamental data types

1. Integer (int)
2. Character (char)
3. Floating point (float)
4. Double precision floating point (float)
5. void
2.4 Data Types

1. Integer Types: Integer are whole number

   for 16 bit machine size of integer value -32768 to +32767

   (i.e. \(-2^{15}\) to \(2^{15} - 1\)). One bit reserved for sign and 15 bits for magnitude of number.

2. Floating Point Types: floating point(real number) are stored in 32 bits with 6 digits of precession. They are defined by keyword float.

   when accuracy provided by float number is not sufficient, we can use double, to determine number. These are known as double precession number

3. Character Type: is used to represent single character. Both signed and unsigned character occupy one byte memory.
2.5 Constants: Constants refers to fixed value that does not change during the execution of program.
2.5 Constants:

Integer constants: sequence of digits

1. Decimal integer: any combination of digits from 0-9
   e.g. 1, 746, 5278

Invalid decimal integer

<table>
<thead>
<tr>
<th>Integer</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>13,445</td>
<td>Commas not allowed</td>
</tr>
<tr>
<td>38.0</td>
<td>Decimal point not allowed</td>
</tr>
<tr>
<td>5 10 15</td>
<td>Space is not allowed</td>
</tr>
<tr>
<td>876-54-578</td>
<td>Special symbol ‘–’ not allowed</td>
</tr>
<tr>
<td>$</td>
<td>Special symbol ‘$’ not allowed</td>
</tr>
</tbody>
</table>
Octal constants: combination of digits taken from 0 to 7. First digit must be zero.
  e.g. 0, 037, 0435

Invalid decimal integer

<table>
<thead>
<tr>
<th>Integer</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>743</td>
<td>First digit is not 0</td>
</tr>
<tr>
<td>04280</td>
<td>Due to number 8</td>
</tr>
<tr>
<td>077.77</td>
<td>Decimal point not allowed</td>
</tr>
</tbody>
</table>
Hexagonal constants: sequence of digits which begins with 0x or 0X is considered as hexagonal. It can be followed by combination of digits taken from 0 through 9 and through f (or F).

e.g. 0x, ox1, 0x9f, 0xbc, ox7fff

Invalid decimal integer

<table>
<thead>
<tr>
<th>Integer</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12.34</td>
<td>Decimal point not allowed</td>
</tr>
<tr>
<td>0BE38</td>
<td>Not begins with 0x or 0X</td>
</tr>
<tr>
<td>0x,4bfff</td>
<td>Commas not allowed</td>
</tr>
<tr>
<td>0XDEFG</td>
<td>Due to presence of G</td>
</tr>
<tr>
<td>0X45H3</td>
<td>Due to presence of H</td>
</tr>
<tr>
<td>Hex3456</td>
<td>Hex not allowed</td>
</tr>
</tbody>
</table>
Real constants (Floating point constant) :
It is a base 10 number that contain either a decimal point or exponent (or both)
e.g. 0.03 0.0 980.012 0.000123 1.2345E+8 0.005e-5

Invalid decimal integer

<table>
<thead>
<tr>
<th>Integer</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Decimal point is not there</td>
</tr>
<tr>
<td>2,000.0</td>
<td>Commas not allowed</td>
</tr>
<tr>
<td>4E+10.2</td>
<td>Decimal point in exponent</td>
</tr>
<tr>
<td>2E 10</td>
<td>Space is not allowed</td>
</tr>
<tr>
<td>1/2</td>
<td>/ is not allowed</td>
</tr>
</tbody>
</table>
Character constants:

**Single Character Constant:** it contains a single character enclosed within single quote mark. It may be single alphabet, a single digit or a special symbol enclosed within single inverted comma.

e.g. ‘a’ ‘5’ ‘\n’

Invalid ‘ab’ ‘54’

**String Constant:** It is sequence of character enclosed in double quote. It consists of any number of consecutive character (including none) enclosed in double quotation mark.

e.g. “green” “Hellow” “$19.94” “X” “First C Program” “?-------” “5+3”
**Backslash character Constant (Escape Sequence)**

Certain Characters can be represented in character and string constants by escape sequence like `\n` (newline);

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>Alert</td>
<td>Sounds a beep</td>
</tr>
<tr>
<td>\b</td>
<td>Back space</td>
<td>Backs up 1 character</td>
</tr>
<tr>
<td>\f</td>
<td>Form feed</td>
<td>Starts a new screen of page</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
<td>Moves to beginning of next line</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
<td>Moves to beginning of current line</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal tab</td>
<td>Moves to next tab position</td>
</tr>
<tr>
<td>\v</td>
<td>Vertical tab</td>
<td>Moves down a fixed amount</td>
</tr>
<tr>
<td>\</td>
<td>Back slash</td>
<td>Prints a back slash</td>
</tr>
<tr>
<td>'</td>
<td>Single quotation</td>
<td>Prints a single quotation</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quotation</td>
<td>Prints a double quotation</td>
</tr>
<tr>
<td>?</td>
<td>Question mark</td>
<td>Prints a question mark</td>
</tr>
</tbody>
</table>

`\0` Null character Represent null character
2.6 Variables and Declaration:

Variable is an identifier which is used to represent some specified information in the program. Variable definition associates a memory location with variable name. variable – its type, its value and its address. Variables – only one value assigned – its value may change during the execution of program.

e.g. f = 1.8*C+32

Here f and C are variables.

Declaration of variable:

_Type variable-list_

Purpose:

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

Primary Type Declaration: _data-type v_1, v_2,…..v_n;

(v_1, v_2,…..v_n are the names of variables. Note that variables are separated by commas)
Variables meaning:

<table>
<thead>
<tr>
<th>Variables</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a,b, c;</td>
<td>a, b, c are declared as integer variables.</td>
</tr>
<tr>
<td>float root1, root2;</td>
<td>Root1 and root2 are floating point variables</td>
</tr>
<tr>
<td>double ratio;</td>
<td>Ratio is double variable.</td>
</tr>
<tr>
<td>char flag;</td>
<td>Define character type flag variable</td>
</tr>
<tr>
<td>text [100]</td>
<td>Square bracket enclosed size of text</td>
</tr>
</tbody>
</table>

Integer type variable can be declared as short integer for smaller integer (2 byte capacity) and long integer for longer integer (8 byte) and for ordinary integer (4 byte).

**Defining Constant:** In C, a value can be assigned to a variable name. If no value is assigned, it is undefined.

For e.g. int a=3, p=-384, val=1234, float y, z=2.5678e-6 in this case y is undefined. Unless variable is defined it cannot be used in arithmetic expression.
2.7 **Symbolic Constant**: the symbolic constant is a name that substitutes for sequence of character. The character may be numeric constant, character constant or string constant. E.g. pi

**Symbolic constant (Constant Identifier) is defined as**

```
#define Symbolic_name Value of constant
```

#define is called as preprocessor directive.

e.g. #define PI 3.14159

```plaintext
#define TAXRATE 0.20
#define TRUE 1
#define PASS_MARK 40
```
The rules applied to a \#define statement which defines Symbolic Constant:

1. Symbolic names are written in capital (no rule but distinguish from normal variable names).

2. No blank space between \# and word define.

3. \# must be first character in line.

4. A blank space is required between \#define and symbolic name and between symbolic name and the constant.

5. \#define statement must not end with semicolon.

6. After definition, the symbolic name should not be assigned any other value within program.

7. Symbolic names are not declared for data types.
2.8 Input and Output Functions: there are two methods of providing data one method to assign values of variables through assignment statements like \( x=1 \), \( y=6 \) and so on. another methods is use of input function. C has library of input and output functions.

**Input functions**: `scanf()`, `getch()`, `gets()` etc.

**Output functions**: `printf()`, `putc()`, `putchar()`, `puts()` etc.
2.8.1 **Scanf() function**: the input data can be entered in the computer by means of `scanf()` function. Using this function we can enter any combination of numerical value, signed character or string.

`scanf()` function is written as:

```
Scanf("control string", arg1, arg2,.....argn);
```

**Control string**: certain required formatting information

**arg1, arg2,…..argn**: individual data items

**Example**:

```c
Int I, j;
Char c;
Float x;

scanf("%d", &i);
scanf("%d %c %f", &j, &c, &x);
```
2.8.1 **Scanf() function:**

- Argument refers to pointer that indicates the address of data items within computer memory. When the variable is declared memory location in the memory is allotted to the variable. The & character is the address of operator in C, it returns the address in memory of the variable it acts on.

- The control string consists of individual group of character, with one character group for each data item.

- Each character group begins with % sign followed by conversion character of data item.

- Within control string multiple character can be separated by white space character (Blank space, tabs, new line.
### 2.8.1 Conversion character for data input

<table>
<thead>
<tr>
<th>Conversion Character</th>
<th>Type of Argument</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
<td>Integer</td>
<td>Signed decimal number</td>
</tr>
<tr>
<td>%i</td>
<td>Integer</td>
<td>Signed decimal integer</td>
</tr>
<tr>
<td>%o</td>
<td>Integer</td>
<td>unsigned Octal number</td>
</tr>
<tr>
<td>%u</td>
<td>Integer</td>
<td>Unsigned decimal integer</td>
</tr>
<tr>
<td>%X</td>
<td>Integer</td>
<td>Unsigned Hexadecimal integer</td>
</tr>
<tr>
<td>%f</td>
<td>Floating point</td>
<td>Signed floating</td>
</tr>
<tr>
<td>%e</td>
<td>Floating point</td>
<td>Signed value in the form of exponent</td>
</tr>
<tr>
<td>%g</td>
<td>Floating point</td>
<td>Signed value in the form of e or f form</td>
</tr>
<tr>
<td>%E</td>
<td>Floating</td>
<td>Same as e</td>
</tr>
<tr>
<td>%G</td>
<td>Floating</td>
<td>Same as g</td>
</tr>
<tr>
<td>%S</td>
<td></td>
<td>Signed string</td>
</tr>
</tbody>
</table>
2.8.2 getchar() function:

- A single character can be entered into the computer using C Library function getchar().

- `character_variable = getchar();`

- When this statement is encountered, a computer waits until the key is pressed, then assigns this character as value to getchar function.

2.8.3 getch() function:

- getch() is similar to getchar(). But it wait for key press, after which it return immediately

- Char c;

- c = getch();
2.8.4 gets() function:

- A function gets receive the string from the standard input device, while puts output the string to standard output device.
- To enter multiword string we have to use gets()
- Space and tabs are acceptable

```c
#include<stdio.h>

main()
{
    printf("Enter name : ");
    scanf("%s", name);
    printf("%s", name);
}
```

Output

Enter name: sachin patil
    sachin
2.8.5 `printf()` function:

- `printf()` function is used for formatted output. It can take simultaneous number of arguments. E.g. `printf("%d %d", l, j);`, `printf("%d %f", l, x);`

<table>
<thead>
<tr>
<th>Conversion Character</th>
<th>Type of Argument</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d</td>
<td>Integer</td>
<td>Signed decimal number</td>
</tr>
<tr>
<td>%i</td>
<td>Integer</td>
<td>Signed decimal integer</td>
</tr>
<tr>
<td>%o</td>
<td>Integer</td>
<td>unsigned Octal number</td>
</tr>
<tr>
<td>%u</td>
<td>Integer</td>
<td>Unsigned decimal integer</td>
</tr>
<tr>
<td>%X</td>
<td>Integer</td>
<td>Unsigned Hexadecimal integer</td>
</tr>
<tr>
<td>%f</td>
<td>Floating point</td>
<td>Signed floating</td>
</tr>
<tr>
<td>%e</td>
<td>Floating point</td>
<td>Signed value in the form of exponent</td>
</tr>
<tr>
<td>%g</td>
<td>Floating point</td>
<td>Signed value in the form of e or f form</td>
</tr>
<tr>
<td>%ld</td>
<td></td>
<td>Signed long</td>
</tr>
<tr>
<td>%lu</td>
<td></td>
<td>Unsigned long</td>
</tr>
<tr>
<td>%lf</td>
<td></td>
<td>Double floating point</td>
</tr>
</tbody>
</table>
2.8.6 putchar() function:

- single character can display using C library function putchar.
- It transmit single character to standard output device (monitor)

putchar written as

Putchar (Character_variable);

Character_variable is type of variable containing a character

e.g

answer= ‘N’;

Putchar(answer);

Character will display N on the screen
2.8.7 puts() function: A function puts() exactly opposite to the gets() function. It outputs a string on screen.

```
#include<stdio.h>

main()
{
    Character Test Function | Meaning
    ----------------------------------------
    isalnumeric(c)          | Is c an alphanumeric character?
    isalpha(c)              | Is c an alphabetic character?
    isdigit(c)              | Is c a digit?
    islower(c)              | Is c a lower case character?
    isupper(c)              | Is c an upper case character?
    isprint(c)              | Is c a printable character?
    ispunct(c)              | Is c a punctuation mark?
    isspace(c)              | Is c a white space character?

Char name [40];
printf("\n enter name :");
gets(name);
Puts(name);
}
```

Output

Enter name: sachinpatil

sachinpatil
2.9 Formatted input and output:

Scanf enter data from standard input device and store in computer memory but printf move data from computer memory to standard memory device

Printf statement format:

```
Printf("control string", arg1, arg2,......,argn);
```

Control string contain formatting information- it consists of individual group of character and must begin with %sign.

Arg1, arg2 … represent the output data item. It may be variables or array name or more complex expression
Output of integer number:

The format specification for printing an integer number is \%wd where, \( w \) specifies minimum field width for output and \( d \) specifies value to be printed in integer. If number is greater than the specified width, it will be printed in full, overriding the minimum specification. The number always right justified in given field width.

e.g. output of integer number \( i = 1234 \) in different format.

<table>
<thead>
<tr>
<th>Format</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>printf(&quot;%d&quot;, i)</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>printf(&quot;%6d&quot;, i)</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>Printf(&quot;%-6d&quot;, i)</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>printf(&quot;%2d&quot;, i)</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>printf(&quot;%06d&quot;, i)</td>
<td>0 0 1 2 3 4</td>
</tr>
</tbody>
</table>
Output of real number

f type conversion and e type conversion are used

Format of specification

%w.pf  or %w.pe

The integer w indicates the minimum number of positions that are to be used for display and p indicates precession (number of digits to be display after decimal point).

e.g. output number of x=12.3456
### e.g. output number of x=12.3456

<table>
<thead>
<tr>
<th>Format</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>printf(\%7.4f&quot;,x)</code></td>
<td>1 2 . 3 4 5 6</td>
</tr>
<tr>
<td><code>printf(\%7.2f&quot;,x)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 2 . 3 5</td>
</tr>
<tr>
<td><code>printf(\%-7.2f&quot;,x)</code></td>
<td>1 2 . 3 4</td>
</tr>
<tr>
<td><code>printf(\f&quot;,x)</code></td>
<td>1 2 . 3 4 5 6</td>
</tr>
<tr>
<td><code>printf(\%10.2e&quot;,x)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 . 2 3 E + 0 1</td>
</tr>
<tr>
<td><code>printf(\%11.4e&quot;,-x)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- 1 . 2 3 4 7 E + 0 1</td>
</tr>
<tr>
<td><code>printf(\e&quot;,x)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 . 2 3 4 5 6 0 E + 0 1</td>
</tr>
</tbody>
</table>
Flag: Each character group within control string can include a flag which affect output. The flag must be placed immediately after % sign.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Data item is left justified within the field. Remaining field will be blank.</td>
</tr>
<tr>
<td>+</td>
<td>A sign (either + or -) will precede the signed numeric data. Without this flag only negative items are preceded by sign.</td>
</tr>
<tr>
<td>0</td>
<td>Cause leading zeros to appear instead of leading blanks. It is applied to data that are right justified within field whose minimum size is larger than data item.</td>
</tr>
<tr>
<td>#(within 0 and x type conversion)</td>
<td>Cause octal and hexadecimal data item to be preceded by 0 and 0x respectively.</td>
</tr>
<tr>
<td># with e, f and g type conversion</td>
<td>Cause decimal point to be present in all floating number even it whole number. It also prevents truncation of trailing zero and g type conversion.</td>
</tr>
<tr>
<td>Blank space</td>
<td>A blank space will precede each +ve signed numeric data item.</td>
</tr>
</tbody>
</table>
When `scanf` is executed, a string of undetermined length (but not more than 79 char) will be entered from keyboard and assigned to `line`.

Notice that there is a blank space preceding `%[^\n]` to ignore any unwanted character that may have been entered previously.
The consecutive non-white character that defines the data item collectively define a field. For integer the field width specification is given by %wd

Where w is integer which specifies field width.e.g.

```
Main()
{
    Int x, y, z;
    ...
    Scanf("%3d%3d%3d", &x,&y,&z);
    ...
}
```

When program is executed, three integer quantity is entered through keyboard as

<table>
<thead>
<tr>
<th>1 2 3</th>
<th>Following assignment will result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>x=1, y=2, z=3</td>
</tr>
<tr>
<td>123 456 789</td>
<td>X=123, y=456, z=789</td>
</tr>
<tr>
<td>123456789</td>
<td>X=123, y=456, z=789</td>
</tr>
<tr>
<td>1234 5678 9</td>
<td>X=123, y=4, z=567</td>
</tr>
</tbody>
</table>
2.10 Operators and Expressions:

Types of operators:

1. Arithmetic Operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Incremental and decremented operators
6. Conditional operators
7. Bitwise operators
8. Special operators
2.10 Operators and Expressions:

1. Arithmetic Operators: are used to perform arithmetic operations.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition or unary plus</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction or unary minus</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulo division (remainder after division) modulus operator</td>
</tr>
</tbody>
</table>

e.g. integer arithmetic
Let a and b be declared by statement:
Int a=17, b=4
Then a+b=21
a-b=13
A*b=68
a/b=4 …..the result is truncated, decimal point is discarded
a%b=1…the result is remainder and sign of the remainder is always sign of the first operand
2.10 Operators and Expressions:

Floating point arithmetic:

% (modulus operator) is not applicable to floating point arithmetic.

e.g. let x and y be declared by statement
Int x=14.0, y=4.0;
Then floating point arithmetic will be
A= x/y = 3.500000
B= y/x = 0.285714
C= x+y=18.000000

2.10.2 Assignment operators: are used to assign the result of an expression to a variable. =

identifier = expression;
identifier represents a variable name and expression represents constant or variable.
e.g.
1. X =3 means integer value 3 is assigned to x
2. Sum= a+b means sum of a and b is assigned to sum
The statement x+ = 5; will add the number 5 to the value of x. similarly a+ = b-c will evaluate b-c first, and adds the result to the value of a.
2.10.2 Assignment operators:

1. If i is int type variable and expression i=3.3, the I will assigned value 3.

2. If i and j are both integer and j has been assigned value 7, then several assignment expression will be

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=j</td>
<td>7</td>
</tr>
<tr>
<td>i = j/2</td>
<td>3</td>
</tr>
<tr>
<td>i= 2*j/2</td>
<td>7 (left to right associatively)</td>
</tr>
<tr>
<td>i=2*(j/2)</td>
<td>6 (truncated division followed by multiplication)</td>
</tr>
</tbody>
</table>

Additional assignment operators: +=, -=, *=, /= and %= (short hand operators)

The assignment expression expr 1+= expr 2

Is equivalent to

expr 1= expr 1+ expr 2

expr 1 *=expr 2

Is equivalent to

expr 1 = expr 1*expr 2

short hand operators are easy to write, more concise and easier to read
2.10.2 Assignment operators:

e.g. suppose i and j are integer variables whose values are 3 and 5, and f and g are floating point variables whose value are 5.5 and -3.25

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent expression</th>
<th>Final value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i+=5</td>
<td>i=i+5</td>
<td>8</td>
</tr>
<tr>
<td>f-=g</td>
<td>F=f-g</td>
<td>8.75</td>
</tr>
<tr>
<td>j*=(i-2)</td>
<td>j=j*(i-2)</td>
<td>5</td>
</tr>
<tr>
<td>f/=3</td>
<td>f=f/3</td>
<td>1.833333</td>
</tr>
<tr>
<td>i%=(j-2)</td>
<td>i=i%(j-2)</td>
<td>0</td>
</tr>
</tbody>
</table>
2.10.3 Increment and decrement Operators:

Increment ++

Decrement –

e.g. i++; /* equivalent to i=i+1; */

++ and – can be used prefix or postfix notation.

In prefix notation ++i the value of the variable i is either incremented and is then read while in postfix notation i++ the value of variable is read first and is then incremented.
2.10.4 Relational operators:

- Relational operators are used to compare arithmetic, logical and character expression.

- We have to compare two quantities and depending upon relation, take some decision. It evaluates to zero(0) if condition false and one(1) if condition is true.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Expression</th>
<th>Interpretation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>i&lt;j</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than equal to</td>
<td>i+j=&gt;k</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>(i+j)&gt;(i+10)</td>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than equal to</td>
<td>K!=6</td>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>==</td>
<td>Is equal to</td>
<td>J= =4</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Equality operators

E.g. if i, j, k are integer variables having values 2, 4, 6 respectively then
2.10.5 logical Operators: It is used when we want to test more than one condition and make decision.

- In case of && operator both condition are true then only corresponding loop will be executed otherwise if one condition is false, the loop will not executed.

- In case of || operator if both conditions are false then only corresponding loop will not be executed.

- if one condition is true, the logical operator will be true.

**e.g.** i is an integer variable whose value is 14, f is a floating point variable whose value is 11.1 and c is a character variable representing the character ‘a’ check the following logical expression

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interpretation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>((i&gt;=12)&amp;&amp;(c=='a'))</td>
<td>True</td>
<td>1</td>
</tr>
<tr>
<td>((i&gt;=12)</td>
<td></td>
<td>(c==97))</td>
</tr>
<tr>
<td>((f&lt;20)&amp;&amp;(i&gt;50))</td>
<td>False</td>
<td>0</td>
</tr>
<tr>
<td>((c!='w')</td>
<td></td>
<td>(i+f&lt;=10))</td>
</tr>
<tr>
<td>!(i&gt;(f+1)))</td>
<td>False</td>
<td>0</td>
</tr>
</tbody>
</table>

**e.g.** main()
{ int i, j = 1;
i = ++j  : i has value 2, j has value 2
i = j++  : I has value 3, j has value 4
}
2.10.6 Conditional operators (ternary operator):

The operator that makes use of condition (?:). It is used in if-else statement.

```
expr1 ? expr2 : expr3; this is equivalent to
if (expr1)
    (expr2);
else
    (expr3)
```

While evaluating, expr1 is evaluated first. If expr1 is true (non zero value) Then expr2 is evaluated and this becomes conditional expression. If expr1 is false, then expr3 is evaluated and this become value of conditional operator.

e.g. 1
Assume that i is integer
(i<0)?0 : 100
The expr (i<0) is evaluated first. If it is true, then entire statement take value 0. otherwise (if i is not less than 0) conditional operator take value 100

e.g. 2
a= 10;
b=15;
c= (a>b)?a:b;
Then c=b=15 will be assign to c
2.11 Iterative Statements  (Loop control Structures):

A segment of program code that is executed repeatedly is called loop. Repetition is done until some condition is satisfied.

1. **Top tested loop or entry controlled loop:** in this loop condition is tested before the loop is executed.

2. **Bottom tested loop (exit controlled loop):** in the bottom tested loop, the condition is tested after the loop is executed.

There are three loop structures in c language, these are:

1. While statement
2. do….while statement
3. For statement
2.11.1 while statement:

It is also called as while loop. It is simplest looping structure. It is often used when the number of times the loop to be executed is not known in advance but depends on test condition. The while is also called as entry controlled loop statement.

The test condition is evaluated first and if test condition is true then the body of loop is executed.

The general format of while statement is:

```plaintext
While (test condition)
{
    Body of loop
}
```

The statement is executed repeatedly as long as expression or test condition is true. The process of repeated execution of body continue until test condition is true. When test condition becomes false, the control is passed out of loop.
e.g. we want to display consecutive digits 0,1,2,3,4,5 with each digit on one line

write c program

C program without using while Statement

```c
#include<stdio.h>
Main()
{
    Int digit=0;
    While (digit<=5);
    {
        Printf("%d\n", digit);
        ++digit:
    } 
}
```

C program using while Statement

```c
#include<stdio.h>
Main()
{
    Int digit=0;
    While (digit<=5)
    { Printf("%d\n", digit++): 
    }
}
```

Output of this program

0
1
2
3
4
5

Dr. Mahesh M. KAMBLE
2.11.1 do while statement:

When while loop is used, the test for continuous of the loop is carried out at the beginning of each pass. Sometimes it is desirable to have a loop with test for continuation at the end of each pass. This can be done by do while statement.

The general form of do-while statement:

```
do
{
    statement
    While ( condition);
}
```

It is natural to test a loop at the beginning rather than at the end of loop. Hence do-while loop is less frequently used than while loop.
e.g. to print first five digits

```c
#include<stdio.h>

Main()
{
    int digit=0;
    do
    {
        printf("%d\n", digit ++);
    }
    While (digit<=5);
}
```
2.11.3 The for statement (for loop):

It is flexible, powerful and most common looping structure used in C. It is useful for repetitive task.

Syntax: for (expr1; expr2; expr3)

where expr1- is used to initiate some parameter that controls looping action.

expr2- represents a condition that must be true for the loop to continue execution.

expr3-is used to alter value of parameter initially assigned by expression1, i.e. expr 1

Here expr1 is assignment statement;
expr2 is logical statement;
expr3 is unary expression or assignment expression.

When for statement is executed, expr2 is evaluated and tested at the beginning of each pass through the loop and expr3 is evaluated at the end of each pass.

Thus for statement is equivalent to

expr1;
While(expr2);
{
Statement;
expr3;
}
The looping action continuous as long as value of expr2 is true. It is top testing loop like while loop. But while loops are used when number of passes is not known in advance and for loops are used when number of passes are known advance.

Equivalence of for and while loop:

| for(e1;e2;e3) | e1;  
| { | While(e2)  
| Statement; | {  
| } | Statement;  
| } | e3;  
| } |}
/*To display the number 0 to 5 using for loop*/

#include<stdio.h>

Main()
{
    Int digit;
    For (digit=0; digit<=5; ++digit)
    { printf("%d\n", digit);
    }
}

Flow chart of for loop

Dr. Mahesh M. KAMBLE
2.11.4 Nesting of for loops:

Nesting of loop is structure in which one for statement is within another for statement.

```
for (i=1; i<5; ++i)
{
    for (i=1; i<5; ++i)
    {
        // Code
    }
}
```
2.12 Decision making statement:

In many situations, we have to test the condition at some point and select alternative paths depending upon the results of the condition. This is known as branching.

There are three decision-making statements:

1. If statement
2. If ….. else statement
3. Switch statement
2.12.1 simple if statement: This is simple form of decision making statement. It allows to make decision after evaluating a test expression. Depending upon the result (true or false), the program execution proceed in one direction.

The general form is

if (test condition)
{
    Statement block;
}
Statement x;

Statement block may be single statement or group of statements. If test condition is true, the statement block will executed, otherwise the statement block will be skipped and the execution will jump to statement x. When test is true, both block statement and statement x will be executed. It is a two way decision statement.
Examples

1. if (x<0)
   printf("%f", x);

   The value of x will be displayed if its value is negative.

2. if (x <=3.0)
   {
     y=3*pow(x,2);
     printf("%f\n", y);
   }

   The value of y is evaluated and displayed if value of x does not exceed 3.
2.12.2 if……else statement: it is extension of simple if statement. The if statement will execute the statement if the expression is true, otherwise it will be skipped. However in many cases we require another statement to be executed if test expression is false. This is possible using if……else statement.

The general form is

```c
if (test expression)
{
   True block statement (s);
}
else
{
   False block statement (s);
}
```

If test expression is true then true block statement will be executed otherwise (i.e. if expr is false), false block statement(s) will be executed. The control is transferred subsequently to statement x.
Examples

1. if (a<b)
   
   printf("a is smaller");

   else

   printf("b is smaller");

2. if (number%2==0)
   
   printf("The number is even");

   else

   printf("The number is odd");
2.12.3 Nesting of if……else statement

when a series of decision are involved, we have to use more than one if…else statement s in nested form. Such nesting provides a programmer with a lot of flexibility in programming.

Nested could be in several form. Some of them are below

1. if (expr1)
   statement 1;
else
   if (expr2)
   statement 2;

2. If (expr1)
   {
   if (expr2)
   Statement 1;
   }
else
   Statement 2;

3. If (expr1)
   {
   if (expr2)
   Statement 1;
   else
   Statement 1;
   }
else
   if (expr3)
   statement 3;
else
   Statement 4;

4. if (expr1)
   If (expr2)
       Statement 1;
   else
       Statement 2;
   else
       if (expr3)
           statement 3;
       else
           Statement 4;

Dr. Mahesh M. KAMBLE
if (Test condition 1)
{
  If (Test condition 2)
  {
    Statement 1;
  }
  else
  {
    Statement 2;
  }
  else
  {
    statement 3;
  }
}

Statement x;
Flow chart of nested if ….else statement
2.12.4 The switch statement

when one alternative is selected from many alternatives, then we can design a program using `if statement` to control selection. When number of alternatives increases the complexity increases. The program becomes more difficult to read and execute. To avoid such confusion C has a multiway decision statement called a `switch`. The switch statement test the value of a given variable against list case value and when match is found, a block of statement associated with that case is executed.
switch (exp) {
    Case value 1: {
        block 1
        break;
    }
    Case value 2: {
        block 2
        break;
    }
    ................
    ................
    Default {
        default block
        break;
    }
    Statement x;
}

The expression results is an integer value or it may be character type. The default statement will be executed if none of the case used matches the value of expression. The break statement causes exit from the switch statement and transferring the control to the statement x.
2.12.4 switch statement

- We can mix integer and character constants in different case of switch.
- Sometimes there may not be any statement in switch.
- If there are multiple statements to be executed in each case, there is no need to enclose these within a pair of braces.
- If we have no default case then program fails through entire switch and continue with next instruction.
- Switch can be replaced by if statement. The disadvantage of switch is we cannot have switch like `case i<=20;`
- The break statement used in switch takes control outside switch.
- It is possible to nest switch statement.
- It is often used in menu selection.
2.12.4 switch statement

Switch

expr = value 1
Block 1

expr = value 2
Block 2

(no match)
(default)

Default Block

Statement x
2.13.1 The break statement:

It is used to terminate loops or to exit from switch. When keyword break is encountered in loop, control automatically passes to first statement after the loop. It can be used within for, while, do-while or switch statement.

It is simply written as

Break;

If break is used in while, do-while or for loop, then control will be immediately transferred out of the loop.
2.13.1 The break statement: Examples:

The loop will continue to execute as long as the current value for floating point variable x does not exceed 100. However, computation will break out of the loop if a negative value of x is detected.
2.13.1 The exit using break statement

while (condition)
{
        ................
        ................
        if (condition)
        break;
        ................
}

Exit from loop

do
{
        ................
        ................
        if (condition)
        break;
        ................
}

Exit from loop

Exit from loop

for (condition)
{
        ................
        ................
        if (error)
        break;
        ................
}

Exit from loop

Exit from loop

for (condition)
{
        ................
        ................
        if (condition)
        break;
        ................
}

Exit from loop

Exit from loop

Dr. Mahesh M. KAMBLE
2.13.2 The continue statement:

The continue statement tells the compiler: 
_SKIP FOLLOWING STATEMENT AND CONTINUE WITH THE NEXT ITERATION_

The format is

\[
\text{Continue;}
\]

```plaintext
case
  | if (----------)
  | continue;
  | ..........  
  | ..........  
  | }
```

```plaintext
do
  | 
  | { 
  | if (--------)
  | continue;
  | ..........  
  | ..........  
  | }
```

```plaintext
for (initialization; test condition; increment)
  | 
  | { 
  | if (--------)
  | continue;
  | ..........  
  | ..........  
  | }
```

Dr. Mahesh M. KAMBLE
2.13.3 The goto statement:

The goto statement is used to branch unconditionally from one point to another in program. The format is: goto label;

goto label;
-------------
forward jump
label';
statement

-------------

label;
statement
-------------
goto label';
backward jump

Applications:

➢ Branching around statement or the group of statement thus bypassing the remainder of the loop during current pass.

➢ Jumping to the end of loop under certain condition.

➢ Jumping completely out of loop
2.13.4 The exit () function:

The exit () causes immediately termination of the entire program. Such force termination is required when error is detected by program.

The general form: exit (n);

The integer n is called as exit status. This value is 0 for a program that terminates normally, and non zero for a program that terminates due to some detected error condition. The exit () is used when some mandatory condition for program execution is not satisfied.

The comma operator (,):

The comma operator is used in conjunction with for statement.

The general form

for (expr1a, expr1b; expr2; expr3)
2.14 Library Functions:

- Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.

- Standard input / output character (i.e. read and write numbers, open and close file etc.)

- Operations on character (convert upper case equivalent to lower case)

- Mathematical operations (trigonometric function, square root)

- Operations on string (i.e. compare string, compare string)

- Information is stored in special file which are supplied to computer. This file accomplished with preprocessor `#include`.

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

`#include<math.h>`

Dr. Mahesh M. KAMBLE
2.14 Commonly used library Functions:

<table>
<thead>
<tr>
<th>MATH OPERATIONS</th>
<th>CHARACTER &amp; STRING MANIPULATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs absolute value</td>
<td>atoi convert string to integer</td>
</tr>
<tr>
<td>acos arc cosine</td>
<td>bsearch binary search of array</td>
</tr>
<tr>
<td>asin arc sine</td>
<td>isalnum detect alphanumeric character</td>
</tr>
<tr>
<td>atan arc tangent</td>
<td>isalpha detect alphabetic character</td>
</tr>
<tr>
<td>atan2 arc tangent of quotient</td>
<td>iscntrl detect control character</td>
</tr>
<tr>
<td>cabsf complex absolute value</td>
<td>isdigit detect decimal digit</td>
</tr>
<tr>
<td>cexpf complex exponential</td>
<td>isgraph detect printable character</td>
</tr>
<tr>
<td>cos cosine</td>
<td>islower detect lowercase character</td>
</tr>
<tr>
<td>cosh hyperbolic cosine</td>
<td>isprint detect printable character</td>
</tr>
<tr>
<td>cot cotangent</td>
<td>ispunct detect punctuation character</td>
</tr>
<tr>
<td>div division</td>
<td>isspace detect whitespace character</td>
</tr>
<tr>
<td>exp exponential</td>
<td>isupper detect uppercase character</td>
</tr>
<tr>
<td>fmod modulus</td>
<td>isxdigit detect hexadecimal digit</td>
</tr>
<tr>
<td>log natural logarithm</td>
<td>memchr find first occurrence of char</td>
</tr>
<tr>
<td>log10 base 10 logarithm</td>
<td>memcpy copy characters</td>
</tr>
<tr>
<td>matadd matrix addition</td>
<td>strcat concatenate strings</td>
</tr>
<tr>
<td>matmul matrix multiplication</td>
<td>strcmp compare strings</td>
</tr>
<tr>
<td>pow raise to a power</td>
<td>strerror get error message</td>
</tr>
<tr>
<td>rand random number generator</td>
<td>strlen string length</td>
</tr>
<tr>
<td>sinh sine</td>
<td>strncmp compare characters</td>
</tr>
<tr>
<td>sinh hyperbolic sine</td>
<td>strrrchr find last occurrence of char</td>
</tr>
<tr>
<td>sqrt square root</td>
<td>strstr find string within string</td>
</tr>
<tr>
<td>srand random number seed</td>
<td>strtonk convert string to tokens</td>
</tr>
<tr>
<td>tan tangent</td>
<td>system sent string to operating system</td>
</tr>
<tr>
<td>tanh hyperbolic tangent</td>
<td>tolower change uppercase to lowercase</td>
</tr>
<tr>
<td></td>
<td>toupper change lowercase to uppercase</td>
</tr>
</tbody>
</table>
THANK YOU!
Introduction:

- Simple data types: int, float, double, char etc.

- Structured data type: Array

- When we want to input number of items of the same type then we define array

- Array is the collection of variables of the same type that are referenced by a common name.

- Specific variables or elements in the array are accessed by means of an index into the array
3.1 Concept of array:

- The program prints the value of x as 20 because when value of 20 is assigned to x and stored in memory location, earlier value 10 is lost. Thus ordinary variables are capable of holding only one value at a time. But there are situations when we want to store more than one value at a time in a single variable.
3.1 Concept of array:

Suppose we want to arrange percentage of marks obtained by 100 different students in ascending order then there are two options:

1. Construct 100 variables to store percentage of marks obtained by 100 different students.

2. Construct one variable capable of storing or holding all 100 values.

It will be easier to handle one variable which can handle 100 variables, hence second option is better. Such subscribed variable is referred as array.

Collection of group of similar elements is called array

Dr. Mahesh M. KAMBLE
3.2 Defining array:

- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- An array is a collection of group of similar elements referred by same name.
- Each array name must be accompanied by a size specification.
- Instead of declaring individual variable, such as num0, num1, ……..num99, we can declare one array variable such as num and use num[0], num[1], and ……..num[99] to represent individual variables. A specific element in an array is accessed by an index.
- All array consist of contiguous memory location. The lowest address corresponds to the first element and highest address to the last element.

First Element

```
```

Last Element

Dr. Mahesh M. KAMBLE
3.2 Defining array:

- For one dimensional array, the size is specified by a positive integer expression enclosed in square bracket.

- If we want to represent four numbers (say 10, 20, 25, 30) in an array variable number, then we may declare the variable number as: `int num[4],`

  where, `int` is type of variable, `num` is name of variable and `4` is an array element.

*In general, one-dimensional array definition may be defined as*

```
storage_class data_type array[expression];
```

Where, storage refers to storage class of array,

- data type refers to type of data,

- array refers array name,

- expression refers to positive valued integer expression which indicate array element.

Example:

1. `int x[10];` states that `x` is an 10 element array.

2. `char text[200];` defines `text` to be an 200 element character array.

3. `static char message [20];` message is defined as static 20 element character array.
3.3 Array initialization:

You can initialize array in C either one by one or using a single statement as follows:

```c
```

The data element are referred as `x[0]`, `x[1]`, `x[2]`,……`x[9]` are shown in figure. The number of value between `{ }` cannot be larger than the number of element that we declare for array between square bracket `[ ]`. 

---

Dr. Mahesh M. KAMBLE
3.3 Array initialization:

Each array element is referred by specifying the array name followed by one or more subscripts with each subscript enclosed in square bracket i.e. \( x[0] = 100, x[1] = 101 \) … \( x[9] = 109 \)

Consider following array definition

\[
\text{int dig[ ] = \{1, 2, 3, 4, 5, 6\} digit of six integer array.}
\]

An element in array is accessed by indexing the array name. this is done by placing the index of the element within square brackets after the name of the array. For example \( \text{int } y = \text{digit}[4]; \)

The above statement will take 5\(^{th}\) element from the array and assign the value to \( y \) variable.

If array is

\[
\text{static x[6] = \{-0.3, 0, 0.25\}}
\]

The \( x[0] = -0.3 \)  \( x[1] = 0 \)  \( x[2] = 0.25 \)

\( x[3] = 0 \)  \( x[4] = 0 \)  \( x[5] = 0 \)
3.3 Array initialization:

- When string constant is assigned to an external or static character array as a part of array definition, the array size is usually omitted. The proper array size will be assigned automatically. This will include provision for the null character `\0` which is automatically added at the end of every string.

- The string of character can be declared as array. Consider following definition.

  ```
  char color[4] = "BLUE"
  ```


  The form `color[4]` is incorrect as `\0` is not included in the array. It should be

  ```
  char [5] = "BLUE" (since for string last character should be `\0`) 
  ```

- A string constant is simply a list of characters within double quotes e.g. "HELLO" with `\0` character being automatically appended at the end by the compiler.

- A string may be initialized as simply as follows

  ```
  char str[7] = "Nirali"
  ```

<table>
<thead>
<tr>
<th>'N'</th>
<th>'i'</th>
<th>'r'</th>
<th>'a'</th>
<th>'l'</th>
<th>'i'</th>
<th>\0</th>
</tr>
</thead>
</table>
3.4 Two dimensional Array:

- Multidimensional array of any dimensions are possible in C but in practice only two or three dimensional array are workable.
- Most common multidimensional array is two dimensional array e.g. the computer display, board game, a mathematical matrix etc.

**Syntax:** `type name [row] [column]`

e.g. 2D array of dimension 2 X 3 is declared as

```
Int det [2] [3]
```

```
det [0] [0] det [0] [1] det [0] [2]
```

A two dimensional array is actually an array of arrays, in the above case an array of two integer arrays (two row) each within three elements and is stored row-wise memory.

Dr. Mahesh M. KAMBLE
3.5 Passing Array element to a function:

Array element can be passed to a function by calling the function by value or by reference. In call by value, we pass value of array element to the function. In call by reference we pass address of array element to the function.
3.6 Pointers: A pointer is a variable that represents the location (rather than value) of data items.

- **Fundamental of pointer:** suppose V is variable that represents some particular data item. The compiler automatically assign memory cell for this data item. The data item can be accessed if one knows locations of first memory cell. There are two ways to write address (location)
  1. Use of address operator
  2. Use of pointer

- the address of V’s memory location can be determined by expression &V where is unary operator called address operator. Let us assign the address of V to another variable PV such that
  
  \[ PV = &V \]

- Now, variable PV is pointer to V since it points the location where V is stored in memory.
The data item represented by V can be accessed by expression \(*PV\), where * is unary operator called indirectional operator on pointer variable.

*PV and V represent same data item.

If we write PV=&V and U=*PV then U and V represent same value.

The pointer variable can assign the address of ordinary variable (i.e. PV =&V)

**Pointer Declaration:** When pointer variables are defined the variable name must be preceded by asterisk (*)

```
data_type *ptvar;
```

Where, Ptvar is the name of pointer variable,

data type refers to data type of pointer,

* (asterisk) tells that variable ptvar is pointer

e.g.

1. Char *P_ch1; means P_ch1 is pointer type character
2. Int far *ifarptr; this declare pointer to int.
3.7 Pointer and one dimension array:

- When an array is declared the compiler allocates a base address and sufficient amount of storage to contain all the elements of array in contiguous memory location.
- The location of first element is base address.
- The size of the data type which the pointer variable refers to, is independent on data.

For example, if the pointer variable points to an integer data type, then it access two byte memory, assuming that an integer occupy two byte. A pointer when incremented, always point to location after skipping the number of bytes required for the data points pointed by it.

For example if we declare an array as:

```c
static int a[5] = {1, 2, 3, 4, 5};
```
3.7 Pointer and one dimension array:

For example if we declare an array as

```c
static int a[5] = {1, 2, 3, 4, 5};
```

If base address of `a` is 1000 and assume that each integer requires two byte the five element will be sorted as follows:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Address</td>
<td>1000</td>
<td>1002</td>
<td>1004</td>
<td>1006</td>
<td>1008</td>
</tr>
</tbody>
</table>

The array `a` is defined as constant pointing to first element `a[0]` and therefore value of `a` is 1000, the location where `a[0]` is stored.

`a = &a[0] = 1000`

If we declare `p` as integer pointer then,

`P = a` this is equivalent to `p = &a[0]`

Dr. Mahesh M. KAMBLE
3.7 Pointer and one dimension array:

Now, we can access every value of x using p++ to move from one element to other.

\[ p = \&a[0] = 1000 \]
\[ p +1 = \&a[1] = 1002 \]
\[ p +2 = \&a[2] = 1004 \]
\[ p +3 = \&a[3] = 1006 \]
\[ p +4 = \&a[4] = 1008 \]

Thus, address of \( a[3] \) = base address + (3*scale factor of integer)

\[ = 1000 + (3*2) = 1006 \]

\(*(p+3)\) gives the value of \( a[3] \).

Thus we can use pointer to access array. Such pointer accessing is faster than array indexing.

Dr. Mahesh M. KAMBLE
Applications of pointer:

1. Pointer can be used to simulate passing parameter by reference, hence arguments can be modified.
2. It provides an alternative method to access array element.
3. It can be used for passing array and string to function.
4. Pointers are efficient for handling complex data structures like linked lists, trees, graphs, etc.
5. Pointer is in dynamic memory allocation (Dynamic memory allocation means allocating memory storage space at runtime). We can allocate and deallocate memory whenever required.
3.8 Array of String: Table of string is called array of string

Example: char name[3][10] = {“Sachin”, “Nirali”, “Max Marks”};

<table>
<thead>
<tr>
<th>S</th>
<th>a</th>
<th>c</th>
<th>h</th>
<th>i</th>
<th>n</th>
<th>\0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>i</td>
<td>r</td>
<td>a</td>
<td>l</td>
<td>i</td>
<td>\0</td>
</tr>
<tr>
<td>M</td>
<td>a</td>
<td>x</td>
<td>M</td>
<td>a</td>
<td>r</td>
<td>k</td>
</tr>
</tbody>
</table>

(total storage require 30 bytes)

We know that all string in the array are not equal length or we can say that all string do not require total 10 spaces. Thus instead of assigning fixed length to each string, we can make a pointer to string of varying length. This is main use of pointer in handling the array of string. Example: char *name[3] = {“Sachin”, “Nirali”, “Max Marks”};

<table>
<thead>
<tr>
<th>S</th>
<th>a</th>
<th>c</th>
<th>h</th>
<th>i</th>
<th>n</th>
<th>\0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>i</td>
<td>r</td>
<td>a</td>
<td>l</td>
<td>i</td>
<td>\0</td>
</tr>
<tr>
<td>M</td>
<td>a</td>
<td>x</td>
<td>M</td>
<td>a</td>
<td>r</td>
<td>k</td>
</tr>
</tbody>
</table>

Here name is the pointer pointing to the array of string with 3 string each containing varying number of characters, the declaration allocates total 24 bytes
THANK YOU!

Dr. Mahesh M. KAMBLE
Chapter 4

User-Defined Functions

Dr. Mahesh M. KAMBLE
PDEA’S Anantrao Pawar College, Pirangut, Mulshi, Pune
Introduction:

- library functions: printf, scanf, puts, gets, exit, cos etc.
- C allows a programmer to define their own functions.
- main() is one of such function where execution of program begins.
- Other functions are executed when they are called directly or indirectly by main().
- The use of user defined functions allows a large program to be split up into small components each of which has unique purpose.
4.1 What is function?:

- Function is an independent or self-contained block of statements that performs a specific, well-defined task and may return a value to the calling program.
- The program developed includes problem solving, designing and coding.
- Designing and writing program of small size is easier.
- If the coding is more than 1000 lines then it will difficult to find errors and debug it. Hence large program is broken into small subprogram.

- Such small, compact and more manageable module is called as a function.
4.1 advantages of functions?:

- Functions are useful for modular or structural programming.
- Functions allows large program to be broken into number of small components which will be useful in debugging process.
- It can be used many times in program and in other program also.
- A function can be called again and again (recursive process)
- It is easier to understand program logic.
- Program development becomes easier.
- Frequently used functions can be put together in a customized library.

Types of functions:

1. **Pre defined functions or Library functions:** These functions are pre written, compiled and placed in library (chapter 2)

2. **User defined functions:** These functions are written by users. User has freedom to choose the name, arguments and data type of function
4.2 defining a function:

- The function definition has two principal components: the first line and the body of function.
- First line of function definition contain the type of specification, followed by function name and set of arguments followed by commas and enclosed in parenthesis.

  In general first line can be written as:

  ```
  data-type  function_name (type1,arg1,  type2, arg2 .....typen, argn)
  ```

  Where, data_type represent data type of item that is returned by function_name.
  type1, type2 ....typen represent data type of arguments arg1, arg2……argn

- The data type are assumed to be integer if they are not shown explicitly.
- The arguments are called formal arguments.
- The remaining part of function definition is compound statement that defines action to be taken by the function. Compound statement is also called body of function.
- The statement may be expression statement, control statement one or more return statements.
- The function name may be any valid C identifier and must be unique in particular program.
4.3 Accessing a function:

- A function can be accessed (or called) by specifying its name followed by list of arguments in parenthesis and separated by commas. Depending upon whether function return value or not, a function can be called in two ways:

1. **Function returning the value**: A function can be called by simply using the function name in a statement

```c
main(0
{ int a;
 a = mul(10,20);
Printf("%d\n", a);
}
int mul (int x, int y)
{ int z;
 z= x*y;
 Return(z);
}
```

When the compiler encounters a function call, the control is transferred to the function `mul(x,y)`. This function is executed line by line and value is returned when return statement is encountered. This value is assigned to `a`. 
4.3 Accessing a function:

2. Function not returning the value: A function that does not return any value may not be used in the expression, but can be called to perform certain task specified in the function. Such function may be called by simply starting their names as independent statements.

```c
#include <stdio.h>
void area ();
Void main();
{
    area();
}
void area ()
{
    float area_square;
    float side;
    Printf("Enter the length of side : ");
    Area_square = side*side;
    printf("Area of square = %f", area_square);
}
```

We have just called function area() in main, we can see that there is no variable or anything specified between the pair of round brackets. That is argument is not passed to function. Now in the prototype definition (void area() ) of the function we can see the return value as void . The void means it does not return anything to the calling function.
The return statement:

- The keyword `return` is used to terminate the execution of function and to return the control to the calling portion of the program.

The return statement is written as

```
return (expression);
```

- The expression is optional. If expression is omitted, the return statement simply causes control to revert back to calling portion of program without any transfer of information.
- Only one expression can be included in return statement.
- Presence of empty return statement is recommended to clarify the logic.
4.4 Function prototype:

```plaintext
datatype name (type1 arg1, type2 arg2, .......typen argn);
```

Where: datatype represents data types of items that is returned by */function, name represents function name,

type1, type2 ......typen represent data types of arg1, arg2......argn

Note that function prototype is not mandatory in
4.5 Recursion:

- Recursion is a special of nesting functions, where a function call itself inside it until some specified conditions has been satisfied.
- The process is used for repetitive computation in which each action is started in terms of previous result.
- We must have certain condition to break out the recursion, otherwise recursion is infinite.

Many iterative (i.e. repetitive) problem can be written in this form,

1. The problem must be written in recursive form.
2. Problem statement must include stopping condition.
4.6 Passing arguments to function:

- Functions are called by their name. If the function is without argument, it can be called directly using its names. But for function with arguments, we have two ways to call them,
  1. Call by value.
  2. Call by reference.

1. **Call by value:** when a single value is passed to a function, via an actual argument, the value of actual argument is *copied* into the function. Hence, value of formal argument can be changed within the function but the value of actual argument within the calling routine will not change. This procedure for passing value of argument to a function is known as **passing values**.

   **Main advantage of call by value** is, it allows single valued actual argument to be written in expression than single variable.

   **Disadvantage:** It does not allow information to be transferred back to the calling portion of program via arguments. Thus, this mechanism is one way of transfer of information
#include<stdio.h>

int add (int a, int b) {
    int result;
    result = a + b;
    return (result);
}

void main() {
    int i, j;
    puts("Enter two integer ");
    scanf("%d %d", &i, &j);
    printf("%d+%d=%d
", i, j, add(i, j));
}


4.6 Call by reference:

- Array arguments are passed differently than single valued data item. If array name is specified as actual argument, then individual array elements are not copied. Instead, the location of array is passed to a function. The variables are stored in memory, hence we can pass location number (called address) of variable to function. This is known as call by reference.

```c
#include<stdio.h>
Void fun (int *p);
int main ()
{
    Int x=20;
    fun (&x);
    printf("%d", x)
}
Void fun (int *p)
{
    *p=*p+10;
}
/*passing address of x as argument */
```

Output is 30
THANK YOU!

Dr. Mahesh M. KAMBLE
Chapter 5

Graphics in C

Dr. Mahesh M. KAMBLE
PDEA’S Anantrao Pawar College, Pirangut, Mulshi, Pune
Introduction:

- Computer graphics are used to draw figures of different shapes.
- The screen divided into small picture cells are called **pixels**.
- The pixels are addressable points in the screen.
- Medium resolution (320 x 200 pixels)
- High resolution (640 x 200 pixels)
- Very High resolution (640 x 350 pixels)
- Super high resolution (1028 x 768 pixels)

Different techniques are used for computer graphics like CAD (Computer Aided Design), CAS(Computer Aided Simulation) and CAM(Computer Aided manufacturing)
5.1 Concepts of Graphics in C:

1. **Header File:** the files `graphics.h` and `graphics.lib` are used for graphics program. `graphics.h` contains the definitions of constants and prototypes of graphics function. The file `graphic.lib` is a library file which contain function definition.

2. **Graphic mode:** there are two modes, namely **text mode** and **graphic mode**. In **text mode** it is possible to display text only in terms of ASCII. In **graphics mode** any type of figure can be display, captured or Animated. Automatic selection of mode can be done by standard function `intigraph()`
5.1 Concepts of Graphics in C:

3. **Graphic drivers:** Graphic drivers are programs which communicate with specific devices. Graphic drivers are applicable only in graphic mode.

4. **Resolution:** The display screen is divided into a number of dots called pixels. If pixels are large in number, the image will be more clear.

5. **DETECT macros:** DETECT is a predefined macro which selects appropriate graphic drivers.

6. **Exiting the graph mode:** To exit from the graph mode, we have to use `closegraph()` function.

7. **Restoring the text mode:** The `restorecrtmode()` is used to restore the screen mode to the setting prior to the graphics setting.
Graphic Initialization:

The proper understanding and manipulation of the graphics, we should know the screen co-ordinates. In screen co-ordinate system, the origin is (top, left) corner. The X-co-ordinate go is increasing from 0 to $X_{\text{max}}$ value from left to right and Y-co-ordinates go is increasing from 0 to $Y_{\text{max}}$ from top to bottom. The $X_{\text{max}}$ and $Y_{\text{max}}$ co-ordinates are decided by the choice of the graph mode and graph driver.

<table>
<thead>
<tr>
<th>(0, 0)</th>
<th>(1, 0)</th>
<th>(2, 0)</th>
<th>(3, 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0, 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0, 3)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(X\text{max}, 0)
5.1 Concepts of Graphics in C:
1. The intigraph() function: it is used for initialization of graphics

   **Syntax:** `intigraph(&gd, &gm, " ");`

   It consists of three arguments
   1. `gd` means grapdriver
   2. `gm` means graphic mode
   3. " " should include the driver path

2. *gaphmode*: if *graphdriver = DETECT intigraph sets *gaphmode to the highest resolution available for detected driver.

3. Pathodriver: it specifies the directory path where intigraph looks for graphics drivers first. If pathodriver is null, the driver files must be in the current directory

4. Auto detect feature: `detectgraph()` function
5.2 Some simple Graphics Commands:

1. **getpixel()**: location of points by using x and y co-ordinates.

   **Syntax**: `getpixel (int x, int y);`

2. **Putpixel()**: plots a point on the screen having the X and Y co-ordinates.

   The pixel colour gives the colour of pixel point on the screen.

   **Syntax**: `putpixel (int x, int y, int_pixel_colour);`

   e.g.: `putpixel (50, 100, RED);`

3. **moveto()**: moves the current point to the new position defined by x and y-co-ordinates.

   **Syntax**: `moveto (int x, int y);`

   e.g.: `moveto (100, 150);`
5.2 Some simple Graphics Commands:

4. line(): draw line from \((X_0, Y_0)\) to \((X, Y)\) point

   **Syntax:** \texttt{line (int }x0\texttt{, int }yo\texttt{, int }x\texttt{, int }y\texttt{);}

   e.g. \texttt{line (10,50,10,100);} draws a vertical line.

5. lineto(): draw line in graphics mode from current position to the new position specified by the parameter \(x\) to \(y\).

   **Syntax:** \texttt{lineto (int }x\texttt{, int }y\texttt{);}

   e.g. \texttt{lineto (100,150);}  

6. Polyline: the number of lines using \texttt{linerel()} function. The \texttt{linerel()} function draws a line in the graphics mode from current position to the new position that specifies distance away \((dx,dy)\), where \(dx\) is the distance along X-axis and \(dy\) is the distance along Y-axis.

   **Syntax:** \texttt{linerel(int }dx\texttt{, int }dy\texttt{);} 

   e.g. \texttt{linerel (70,170);}
5.2 Some simple Graphics Commands:

7. **Circle:** The function circle() gives the circle on the screen with the centre $(x,y)$ and having radius defines int radius.

   **Syntax:** `circle(int x, int y, int radius);`

   e.g. `circle(200,100,75);`

8. **Arc:** draws a circular arc having the co-ordinates x and y at centre. The starting angle given by `st_angle` and ending angle given by `end_angle`. The radius of arc is defined by int radius.

   **Syntax:** `arc(int x, int y, int st_angle, int end_angle, int radius);`

   e.g.: `arc(100, 100,0, 90, 30);`
5.2 Some simple Graphics Commands:

9. Ellipse: function draw ellipse on the screen. This ellipse having centre (x,y). The st_angle and end_angle gives the starting and ending angle. The x-radius and y-radius gives semi major and semi minor axis of ellipse.

**Syntax:** `ellipse(int x, int y, int st_angle, int end_angle, int x_radius, int y_radius);`

e.g.: `ellipse (150, 100, 0, 360, 75, 50);`

10. rectangle: the function rectangle() draws a rectangle on the screen using the current drawing colour and thickness of line. The first two points i.e. left and top points gives the one corner points and remaining two points i.e. right and bottom points gives the other points

**Syntax:** `rectangle (int left, int top, int right, int bottom);`

e.g.: `rectangle (100, 50, 250, 150);`
5.2 Some simple Graphics Commands:

11. Bar: the bar() function draws the bar with corner points \((x,y)\) and \((x_1,y_1)\)

   **Syntax:** \(\text{bar}(\text{int } x, \text{int } y, \text{int } x_1, \text{int } y_1,);\)

   e.g. \(\text{bar}(100, 50, 150, 250);\)

12. Polygon: the drawpoly() function draws the outline of polygon. The number of points to be joined and an array giving viewport relative to c0-ordinates.

   **Syntax:** \(\text{drawpoly}(\text{no_points, int_polypoints});\)

   e.g.: int triangle [] = (320, 100, 100, 250, 520, 250, 320, 100);

   \(\text{drawpoly}(4, \text{triangle});\)
5.2 Some simple Graphics Commands:

13. **getmax x and getmax y function**: the function `getmax()` gives the maximum value of x in the current graphic mode. The function `getmax y ()` gives the maximum value of y in the current graphics mode.

**Syntax**: 

\[
\begin{align*}
x1 &= \text{getmax x } () ; \\
y1 &= \text{getmax y } () ;
\end{align*}
\]

Where x1 and y1 be the maximum value of x and y co-ordinates.

14. **Closegraph() function**: This function close the graphics system

**Syntax**: `closegraph();`

15. **Clear device()**: This function clear the screen in graphics mode
5.2 Some simple Graphics Commands:

16. **Setcolour():** this function sets the drawing colour, as specified by variable color.

**Syntax:** setcolor (int color);

- e.g. setcolor (4); /* sets color to red */

The setcolor function is used to set the foreground color in graphics mode. The possible color values are from 0-15. The values with colr are 0-BLACK, 1-BLUE, 2-GREEN, 3-CYAN, 4-RED…

17. **Filling Image:** The set fill style function is used to fill pattern and color:

**Syntax:** void setfillstyle (int pattern, int color);

Following are some predefined fill style:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY_FILL</td>
<td>0</td>
<td>Fill with background color</td>
</tr>
<tr>
<td>SOLID_FILL</td>
<td>1</td>
<td>Solid Fill</td>
</tr>
<tr>
<td>LINE_FILL</td>
<td>2</td>
<td>Fill with _</td>
</tr>
<tr>
<td>SLASH_FILL</td>
<td>3</td>
<td>Fill with //</td>
</tr>
</tbody>
</table>
REFERENCES

“A Book of Computational Physics” (For T.Y.B.Sc. Physics (PH 335), Semester III, as per SPPU syllabus) by Dr. P. S. Tambade, Dr. S. D. Aghav, Dr. V. D. Kulkurni, V. K. Dhas, B. M. Laware: Nirali Prakashan

Let us C : Yashwant Kanetkar

Internet for figures
THANK YOU!

Dr. Mahesh M. KAMBLE