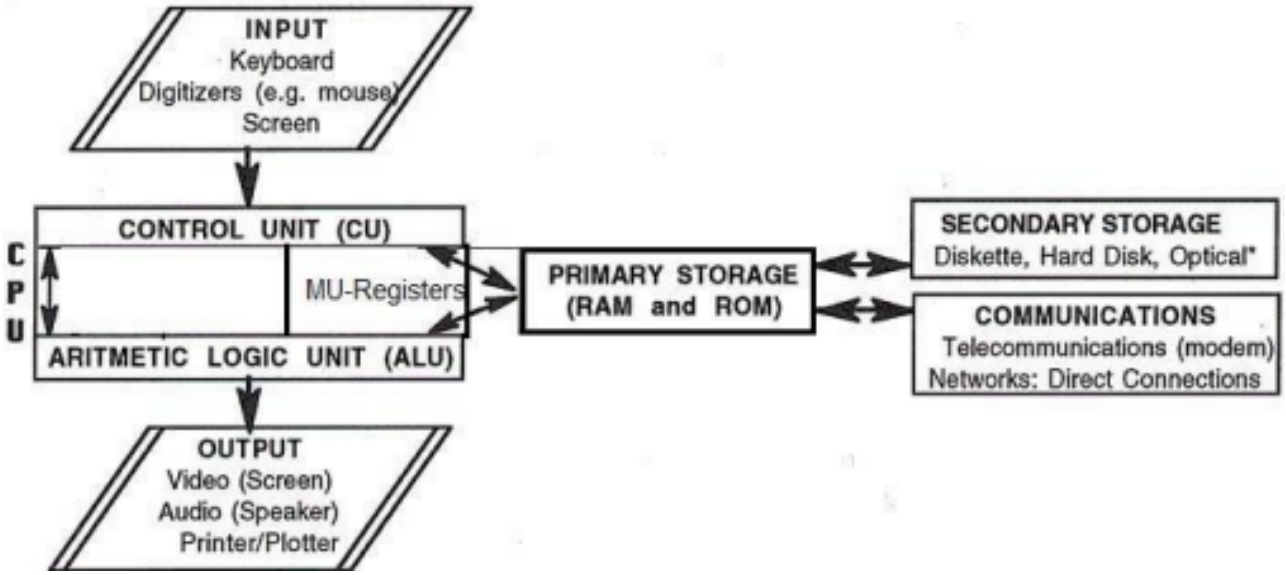
Solution For Introduction to Programming in C

VTU QP_BESCK204E_JUNE/JULY(2023-2024)		
	MODULE 1	
1.a.	Define Computer. With a neat block diagram, explain different components of a computer.	[10]
b.	Explain input and output devices.	[10]
	OR	
2. a.	Explain the following with neat syntax printf() and scanf() functions.	[8]
b.	Define variables. Explain the rules for declaring the variables.	[6]
c.	Explain the structure of 'C' Program.	[6]
	MODULE 2	
3.a.	What are iterative statements? Explain them with neat syntax.	[10]
b.	Write a C Program to find Mechanical Energy of a particle using E = mgh+1/2 mv2	[6]
c.	Explain the uses of goto statement with examples.	[4]
	OR	
4. a.	Explain Relational operators in C language with examples.	[6]
b.	With proper syntax, explain different conditional branching statements. Give suitable examples for each.	[8]
c.	Explain Type Conversion and Type Casting.	[6]
	MODULE 3	
5.a.	With neat syntax, explain function declaration and Function definition.	[6]
b.	Explain the different types of storage classes.	[8]
c.	What is recursion? Write a C program to find the factorial of a number using a recursion function.	[6]
	OR	
6. a.	What is an array? Explain how one dimensional arrays are declared and initialized. Write a C program to find the largest of 'N' elements.	[12]
b.	Write a C program to sort the given set of N numbers using Bubble Sort technique.	[8]
	MODULE 4	
7. a.	List the applications of arrays.	[4]
b.	Write a C program to implement Matrix multiplication and validate the rules of multiplication.	[10]
c.	With syntax and examples, explain the scan set function.	[6]

Mrs. Rajeshwari R, Assistant professor,CSE Dept

	OR	
8. a.	Explain the different methods of reading and writing strings using formatted and unformatted functions. Write an example for each.	[12]
b.	Write a C program to pass a two dimensional array to the function and display in matrix format.	[8]
	MODULE 5	
9. a.	Explain the following string manipulation functions: (i) strlen() (ii) strcpy() (iii) strcmp() (iv) strcat()	[8]
b.	Define pointer. Explain the declaration and initialization of a pointer variable with an example.	[4]
c.	Write a C program to compute the sum mean and standard deviation of all elements stored in an array of N real numbers using pointers.	[8]
	OR	
10. a.	Define structure. Explain the declaration of structure with an example.	[8]
b.	Write a C program to implement structure to read, write and compute average marks and the students scoring above and below the average marks for a class of N students.	[12]

	SOLUTION FOR VTU QP_BESCK204E_JUNE/JULY(2023-2024)	
	MODULE 1	
1.a.	Define Computer. With a neat block diagram, explain different components of a computer. A computer is **an electronic device that can store, manipulate, and process data according to a set of** instructions.  Input unit: The input unit that links the external environment to input data & tasks with the computer system to execute. Data are entered in different forms through different input devices. Keyboard is used for characters input. Mouse is used in GUI (Graphic User Interface). Internally data is processed in machine readable form. Output Unit: Output/result is displayed, printed & transmitted to the outside world. There are many output devices: monitor, printer/plotters, display boards, speakers etc. Storage unit: The data and instructions that are entered into the computer system through input units have to be stored inside the computer before the actual processing starts. Similarly, the results produced by the computer after processing must also be kept somewhere inside the computer system before being passed on to the output units. The storage unit is Primary Memory (RAM) & Secondary (permanent storage devices: disks, tapes) CPU (Central processing Unit): It is the main unit which controls all events within the computer. The CPU has 3 internal units: CU, ALU & Registers:	[10]

CU(Control unit): By selecting, interpreting, and seeing to the execution of the program instructions, the control unit is able to maintain order and direct the operation of the entire system. The control unit acts as a central nervous system for the other components of the computer. It manages and coordinates the entire computer system. It obtains instructions from the program stored in main memory, interprets the instructions, and issues signals that cause other units of the system to execute them.

ALU (Arithmetic & Logic Unit): The arithmetic and logic unit (ALU) is the part where actual computations take place. It consists of circuits that perform arithmetic operations (e.g. addition, subtraction, multiplication, division over data received from memory and are capable of comparing numbers (less than, equal to, or greater than).

MU (Memory Unit/Registers): Registers are built-in memory with CPU having less storage space in bits. Registers are a group of cells used for memory addressing, data manipulation and processing. Instruction Registers, Address registers, Program Counters, Accumulators are examples of registers. ALU takes data from here inside the CPU.

RAM(Random Access Memory): RAM is the memory - primary storage where our data & programs are stored temporarily. It is volatile in nature. After switching off the system everything will vanish from RAM.

ROM(Read Only Memory): ROM is storage medium/"firmware" where some code of the manufacturer is permanently hardwired in the chip which always executes automatically when we start the system. The process is known as POST(Power on Self test). Booting precedes POST.

b. Explain input and output devices. [10]

Input Device Definition: A piece of equipment/hardware which helps us enter data into a computer is called an input device. For example keyboard, mouse, etc.

Output Device Definition: A piece of equipment/hardware which gives out the result of the entered input, once it is processed (i.e. converts data from machine language to a human-understandable language), is called an output device. For example printers, monitors, etc.

List of Input Devices

Given below is the list of the most common input devices along with brief information about each of them.

1. **Keyboard**
- A simple device comprising keys and each key denotes either an alphabet, number or number commands which can be given to a computer for various actions to be performed
- It has a modified version of typewriter keys
- The keyboard is an essential input device and computer and laptops both use keyboards to give commands to the computer

2. **Mouse**
- It is also known as a pointing device
- Using mouse we can directly click on the various icons present on the system and open up various files and programs
- A mouse comprises 3 buttons on the top and one trackball at the bottom which helps in selecting and moving the mouse around, respectively
- In case of laptops, the touchpad is given as a replacement of the mouse which helps in the movement of the mouse pointer

3. **Joy Stick**
- It is a device which comprises a stick which is attached at an angle to the base so that it can be moved and controlled
- Mostly used to control the movement in video games
- Apart from a computer system, a joystick is also used in the cockpit of an aeroplane, wheelchairs, cranes, trucks, etc. to operate them well

4. **Light Pen**
- It is a wand-like looking device which can directly be moved over the device's screen
- It is light-sensitive
- Used in conjunction with computer's cathode ray tube

5. **Microphone**
- Using a microphone, sound can be stored in a device in its digital form
- It converts sound into an electrical signal
- To record or reproduce a sound created using a microphone, it needs to be connected with an amplifier

6. **Scanner**
- This device can scan images or text and convert it into a digital signal
- When we place any piece of a document on a scanner, it converts it into a digital signal and displays it on the computer screen

7. **Barcode Reader**
- It is a kind of an optical scanner
- It can read bar codes
- A source of light is passed through a bar code, and its aspects and details are displayed on the screen

List of Output Device

The commonly used output devices have been listed below with a brief summary of what their function is and how they can be used.

	1. **Monitor** ● The device which displays all the icons, text, images, etc. over a screen is called the Monitor ● When we ask the computer to perform an action, the result of that action is displayed on the monitor ● Various types of monitors have also been developed over the years 2. **Printer** ● A device which makes a copy of the pictorial or textual content, usually over a paper is called a printer ● For example, an author types the entire book on his/her computer and later gets a print out of it, which is in the form of paper and is later published ● Multiple types of printers are also available in the market, which can serve different purposes 3. **Speakers** ● A device through which we can listen to a sound as an outcome of what we command a computer to do is called a speaker ● Speakers are attached with a computer system and also are a hardware device which can be attached separately ● With the advancement in technology, speakers are now available which are wireless and can be connected using BlueTooth or other applications 4. **Projector** ● An optical device which presents an image or moving images onto a projection screen is called a projector ● Most commonly these projectors are used in auditoriums and movie theaters for the display of the videos or lighting ● If a projector is connected to a computer, then the image/video displayed on the screen is the same as the one displayed on the computer screen 5. **Headphones** ● They perform the same function as a speaker, the only difference is the frequency of sound ● Using speakers, the sound can be heard over a larger area and using headphones, the sound is only audible to the person using them ● Also known as earphones or headset	
	OR	
2.a	Explain the following with neat syntax printf() and scanf() functions. Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. Formatted I/O functions 1. printf() 2. scanf() 1. printf(): printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file). Syntax :	[8]

To display any variable value.
printf("Format Specifier", var1, var2, …., varn);
Example:
printf("Enter the text which you want to display");
Output:
Enter the text which you want to display

Or

```
#include<stdio.h>
void main()
{
int a;
a = 20; // Assigning a value in a variable
printf("%d", a); // Printing the value of a variable
}
```

Output:
20

2. scanf():
scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file).

Syntax:
scanf("Format Specifier", &var1, &var2, …., &varn);

Example:

```
#include<stdio.h>
int main()
{
int num1;
printf("Enter a integer number:");
scanf("%d", &num1);
printf("You have entered %d", num1);
return 0;
}
```

Output:
Enter a integer number: 56
You have entered 56

b. Define variables. Explain the rules for declaring the variables. [6]

Definition:

Variables are containers for storing data values, like numbers and characters.

Rules for Naming a Variable in C

We give a variable a meaningful name when we create it. Here are the rules that we must follow when naming it:

1. The name of the variable must not begin with a digit.

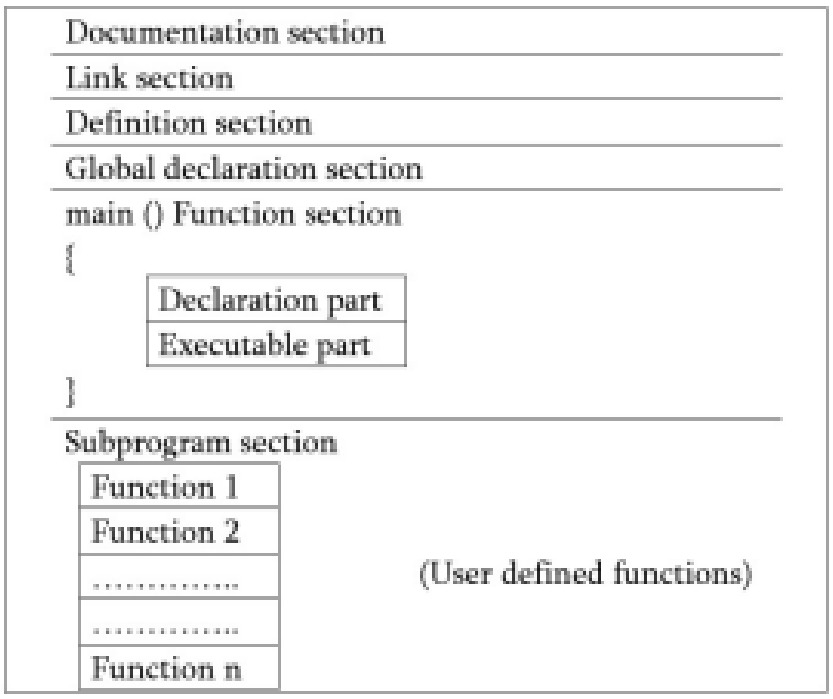
2. A variable name can consist of digits, alphabets, and even special symbols such as an underscore(_).

3. A variable name must not have any keywords, for instance, float, int, etc.

4. There must be no spaces or blanks in the variable name.

5. The C language treats lowercase and uppercase very differently, as it is case sensitive. Usually, we keep the name of the variable in the lower case.

c. Explain the structure of 'C' Program. [6]



Documentation Section: This section is used to write Problem, file name, developer, date etc in comment lines within /*….*/ or separate line comments may start with // . Compiler ignores this section. Documentation enhances the readability of a program.

Link section : To include header and library files whose in-built functions are to be used. Linker also required these files to build a program executable. Files are included with directive #include
Definition section: To define macros and symbolic constants by preprocessor directive #define
Global section: to declare global variables – to be accessed by all functions
main() is the user defined function which is recognized by the compiler first. So, all C program must have user defined
function main() { ……. }. It should have declaration part first then executable part.
Sub program section: There may be other user defined functions to perform specific task when called.

```
/* Example: a program to find area of a circle – area.c
- Documentation Section*/

#include <stdio.h> /* - Link/Header Section */

#define PI 3.14 /* definition/global section*/

int main() /* main function section */
{
 float r, area; /* declaration part */

 print("Enter radius of the circle : "); /* Execution part*/
 scanf("%f", &r);
 area=PI*r*r; /* using symbolic constant PI */
 print("Area of circle = %0.3f square unit\n", area);
 return (0);
}
```

MODULE 2

3.a. What are iterative statements? Explain them with neat syntax. [10]

Iteration is a fundamental concept in programming that involves repeating a specific set of instructions multiple times until a certain condition is met. In the C programming language, there are three types of iteration statements: ***for, while***, and ***do-while***.

For Loop

The ***for loop*** is used to execute a set of statements repeatedly for a fixed number of times. It consists of three parts: ***initialization, condition***, and ***increment/decrement***.

Syntax:

for (initialization; condition; increment/decrement) {

```
  // code to be executed

}
```

The ***initialization statement*** is executed only once at the beginning of the loop, and it is used to initialize the loop variable. The condition statement is evaluated at the beginning of each iteration, and if it is ***true***, the code inside the loop is executed. The ***increment/decrement*** statement is executed at the end of each iteration, and it is used to update the loop variable.

Example of a ***for loop*** that prints the numbers from ***1 to 5*****:**

```
#include <stdio.h>
int main() {
  int i;
  for (i = 1; i<= 5; i++) {
printf("%d\n", i);
  }
  return 0;
}
```

Output:

```
1
2
3
4
5
```

While Loop

The ***while loop*** is used to execute a set of statements repeatedly as long as a certain condition is ***true***.

Syntax:

```
while (condition) {

  // code to be executed

}
```

The ***condition*** is evaluated at the beginning of each iteration, and if it is ***true***, the code inside the loop is executed. The ***loop continues*** until the condition becomes false.

Example of a while loop that prints the numbers from ***1 to 5*****:**

```
#include <stdio.h>

int main() {

  int i = 1;

  while (i<= 5) {

printf("%d\n", i);

i++;

  }

  return 0;

}
```

Output:

```
1
2
3
4
5
```

Do-While Loop

The ***do-while loop*** is used to execute a set of statements repeatedly as long as a certain condition is ***true***. The difference between the ***while loop*** and the ***do-while loop*** is that the ***do-while loop*** executes the code inside the loop at least once before checking the condition.

Syntax:

```
do {

   // code to be executed

} while (condition);
```

The code inside the loop is executed ***first***, and then the condition is checked. If the condition is ***true***, the ***loop continues***, otherwise, it ***terminates***.

Example of a do-while loop that prints the numbers from 1 to 5:

```
#include <stdio.h>

int main() {

  int i = 1;

  do {

printf("%d\n", i);

i++;

  } while (i<= 5);

  return 0;

}
```

Output:

1
2
3
4
5

b. Write a C Program to find Mechanical Energy of a particle using E = mgh+1/2 mv2 [6]

```
#include <stdio.h>
int main(void)
{
float m,h,v,p,k,e;
printf("Enter Mass of the body\n");
scanf("%f",&m );
printf("Enter displacement of the body\n");
scanf("%f",&h );
printf("Enter velocity of the body\n");
scanf("%f",&v );
p=m*9.8*h; //To calculate Potential energy
k=0.5*m*(v*v); //To calculate Kinetic energy
e=p+k;
printf("Potential energy of the body = %f\n",p );
```

```
printf("Kinetic energy of the body = %f\n",k );
printf("Mechanical energy of a body = %f\n" , e);
}
```

Output

```
Enter Mass of the body: 1000.00
Enter displacement of the body: 10.00
Enter velocity of the body; 120.00
Potential energy of the body = 98000.00
Kinetic energy of the body = 7200000.00
Mechanical energy of a body = 7298000.00
```

[4]

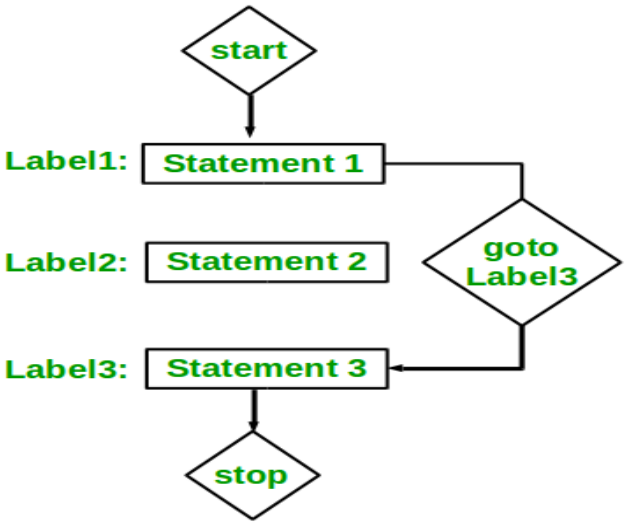
c. Explain the uses of goto statement with examples.

The **C goto statement** is a jump statement which is sometimes also referred to as an **unconditional jump** statement. The goto statement can be used to jump from anywhere to anywhere within a function.

Syntax:

```
Syntax1      |  Syntax2
--------------------------
goto label;  |   label:
.            |   .
.            |   .
.            |   .
label:       |   goto label;
```

In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here, the label is a user-defined identifier that indicates the target statement. The statement immediately followed after 'label:' is the destination statement. The 'label:' can also appear before the 'goto label;' statement in the above syntax.



Flowchart of goto Statement in C

Example:

```
// C program to check if a number is
// even or not using goto statement
#include <stdio.h>

// function to check even or not
void checkEvenOrNot(int num)
{
        if (num % 2 == 0)
                // jump to even
                goto even;
        else
                // jump to odd
                goto odd;

even:
        printf("%d is even", num);
        // return if even
        return;
odd:
        printf("%d is odd", num);
}

int main()
```

```
{
        int num = 26;
        checkEvenOrNot(num);
        return 0;
}
```

Output:
26 is even

OR

4.a. Explain Relational operators in C language with examples. [6]

Relational operators are the symbols that are used for comparison between two values to understand the type of relationship a pair of numbers shares. The result that we get after the relational operation is a boolean value, that tells whether the comparison is true or false. Relational operators are mainly used in conditional statements and loops to check the conditions in C programming.

Relational Operators in C

Operator Symbol	Operator Name
==	Equal To
!=	Not Equal To
<	Less Than
>	Greater Than
<=	Less Than Equal To
>=	Greater Than Equal To

Example:

```
// C program to demonstrate working of relational operators
#include <stdio.h>

int main()
{
        int a = 10, b = 4;

        // greater than example
        if (a > b)
                printf("a is greater than b\n");
        else
                printf("a is less than or equal to b\n");

        // greater than equal to
        if (a >= b)
                printf("a is greater than or equal to b\n");
```

```
        else
                printf("a is lesser than b\n");

        // less than example
        if (a < b)
                printf("a is less than b\n");
        else
                printf("a is greater than or equal to b\n");

        // lesser than equal to
        if (a <= b)
                printf("a is lesser than or equal to b\n");
        else
                printf("a is greater than b\n");

        // equal to
        if (a == b)
                printf("a is equal to b\n");
        else
                printf("a and b are not equal\n");

        // not equal to
        if (a != b)
                printf("a is not equal to b\n");
        else
                printf("a is equal b\n");

        return 0;
}
```

Output:

```
a is greater than b
a is greater than or equal to b
a is greater than or equal to b
a is greater than b
a and b are not equal
a is not equal to b
```

b. With proper syntax, explain different conditional branching statements. Give suitable examples for each. [8]

Conditional Branching Statements in C are used to execute the specific blocks of code on the basis of some condition (as per requirements). Conditional Branching Statements in C are:

- if Statement
- if else Statement
- else-if Statement
- switch Statement

if Statement

This statement is used to execute the specific block of code if a certain condition is evaluated to be true.

Syntax

```
if (condition) {
   //code to be executed if condition specified evaluates is true
}
```

Example

```
#include <stdio.h>
int main() {
   int x = 10;
   if (x > 5) {
      printf("x is greater than 5");
   }
   return 0;
}
```

Output

x is greater than 5

if else Statement

The if-else statement is a decision-making statement that is used to decide whether the part of the code will be executed or not based on the specified condition (test expression).

Syntax

```
if (condition) {
```

```
    // statements
} else{
    // code executed if condition is false
}
```

Example

```
#include <stdio.h>
int main() {
  int x = 10;
  if (x < 5) {
     printf("x is less than 5\n");
  } else {
     printf("x is greater than or equal to 5\n");
  }
  return 0;
}
```

Output

x is greater than or equal to 5

else if Statement

The else if statement in Cis used when we want to check multiple conditions. It follows an "if" statement and is executed if the previous "if" statement's condition is false.

Syntax

```
if (condition1) {
    // code to be executed if condition1 is true
}
else if (condition2) {
    // code to be executed if condition2 is true and condition1 is false
}
```

Example

```
#include <stdio.h>
int main() {
   int x = 10;
   if (x > 15) {
      printf("x is greater than 15");
   }
   else if (x > 5) {
      printf("x is greater than 5 but less than or equal to 15");
```

```
  }
  else {
    printf("x is less than or equal to 5");
  }
  return 0;
}
```

Output

x is greater than 5 but less than or equal to 15

switch Statement

The switch statement in C is used when we have multiple conditions to check. It is often used as an alternative to multiple "if" and "else if" statements.

Syntax

```
switch (expression) {
   case value1:
      // code to be executed if expression equals value1
      break;
   case value2:
      // code to be executed if expression equals value2
      break;
   ...
   default:
      // code to be executed if none of the cases match
      break;
}
```

Example

```
#include <stdio.h>
int main() {
  int x = 2;
  switch (x) {
     case 1:
        printf("x is 1");
        break;
     case 2:
        printf("x is 2");
        break;
     case 3:
        printf("x is 3");
        break;
     default:
```

```
        printf("x is not 1, 2, or 3");
        break;
    }
    return 0;
}
```

Output

x is 2

[6]

c. Explain Type Conversion and Type Casting.

Type Casting: In typing casting, a data type is converted into another data type by the programmer using the casting operator during the program design. In typing casting, the destination data type may be smaller than the source data type when converting the data type to another data type, that's why it is also called narrowing conversion.

Syntax/Declaration:-

destination_datatype = (target_datatype)variable;
(): is a casting operator.
target_datatype: is a data type in which we want to convert the source data type.

Type Casting example –

```
float x;
byte y;
...
...
y=(byte)x;  //Line 5
```

In Line 5: you can see that, we are converting **float(source) data type** into **byte(target) data type**.

2. Type conversion : In type conversion, a data type is automatically converted into another data type by a compiler at the compiler time. In type conversion, the destination data type cannot be smaller than the source data type, that's why it is also called widening conversion. One more important thing is that it can only be applied to compatible data types.

Type Conversion example –

```
int x=30;
float y;
y=x;  // y==30.000000.
```

MODULE 3

5.a. With neat syntax, explain function declaration and Function definition. [6]

Function Declarations
In a function declaration, we must provide the function name, its return type, and the number and type of its parameters.

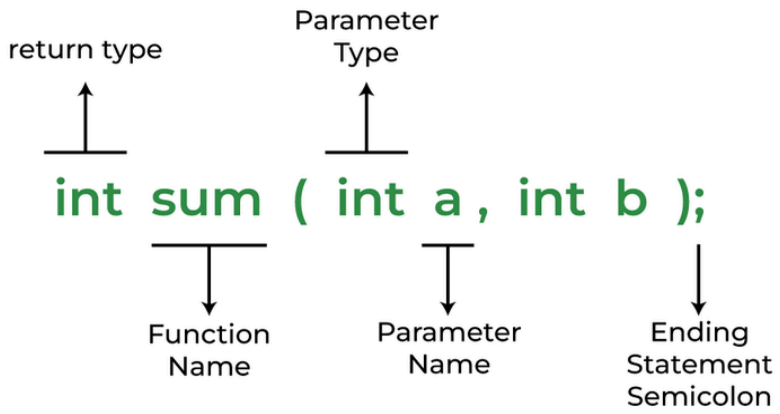
Syntax

return_type name_of_the_function (parameter_1, parameter_2);

The parameter name is not mandatory while declaring functions. We can also declare the function without using the name of the data variables.

Example

```
int sum(int a, int b);  // Function declaration with parameter names
int sum(int , int);     // Function declaration without parameter names
```

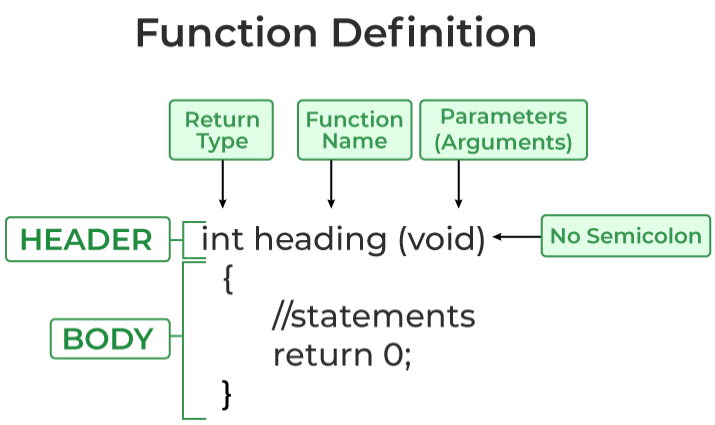


Function Definition
The function definition consists of actual statements which are executed when the function is called (i.e. when the program control comes to the function).

A 'C' function is generally defined and declared in a single step because the function definition always starts with the function declaration so we do not need to declare it explicitly.

SYNTAX:

```
return_type function_name (para1_type para1_name, para2_type para2_name)
{
  // body of the function
}
```



EXAMPLE

```
// C program to show function
// call and definition
#include <stdio.h>

// Function that takes two parameters
// a and b as inputs and returns
// their sum
#include <stdio.h>

int sum(int a, int b)
{
  return a + b;
}

int main()
{
  // Calling sum function and
  // storing its value in add variable
  int add = sum(10, 30);

  printf("Sum is: %d", add);
  return 0;
}
```

Output

Sum is: 40

b. Explain the different types of storage classes. [8]

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

- Automatic
- External
- Static
- Register

Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Till the end of the main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within the function

Automatic

- Automatic variables are allocated memory automatically at runtime.
- The visibility of the automatic variables is limited to the block in which they are defined.
- The scope of the automatic variables is limited to the block in which they are defined.The automatic variables are initialized to garbage by default.
- The memory assigned to automatic variables gets freed upon exiting from the block.
- The keyword used for defining automatic variables is auto.
- Every local variable is automatic in C by default.

Example

```
#include <stdio.h>

int main()

{

int a; //auto
```

char b;

float c;

int d = 10;//auto

printf("%d ",++d);

printf("%d %c %f",a,b,c); // printing initial default value of automatic variables a, b, and c.

return 0;

}

Output:

11 garbage garbage garbage

Static

- The variables defined as static specifier can hold their value between the multiple function calls.
- Static local variables are visible only to the function or the block in which they are defined.
- A same static variable can be declared many times but can be assigned at only one time.
- Default initial value of the static integral variable is 0 otherwise null.
- The visibility of the static global variable is limited to the file in which it has declared.
- The keyword used to define static variable is static.

Example

#include<stdio.h>

void sum()

{

static int a = 10;

static int b = 24;

printf("%d %d \n",a,b);

a++;

b++;

```
}

void main()

{

int i;

for(i = 0; i< 3; i++)

{

sum(); // The static variables holds their value between multiple function calls.

}

}
```

Output:

```
10 24
11 25
12 26
```

Register

- The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.
- We can not dereference the register variables, i.e., we can not use &operator for the register variable.
- The access time of the register variables is faster than the automatic variables.
- The initial default value of the register local variables is 0.
- The register keyword is used for the variable which should be stored in the CPU register. However, it is compiler?s choice whether or not; the variables can be stored in the register.
- We can store pointers into the register, i.e., a register can store the address of a variable.
- Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

Example

#include <stdio.h>

int main()

{

register int a; // variable a is allocated memory in the CPU register. The initial default value of a is 0.

printf("%d",a);

}

Output:

0

External

- The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
- The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.
- The default initial value of external integral type is 0 otherwise null.
- We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
- An external variable can be declared many times but can be initialized at only once.
- If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

Example

#include <stdio.h>

int main()

{

extern int a; // Compiler will search here for a variable a defined and initialized somewhere in the pogram or not.

printf("%d",a);

}

int a = 20;

Output

20

c. What is recursion? Write a C program to find the factorial of a number using a recursion function. [6]

Definition:
Recursion is the process of a function calling itself repeatedly till the given condition is satisfied. A function that calls itself directly or indirectly is called a recursive function and such kind of function calls are called recursive calls.

```
#include <stdio.h>

// A recursive function to calculate factorial
int factorial(int n) {
   if (n == 0) {
      return 1; // Base case
   } else {
      //function calling itself with modified argument
      return n * factorial(n - 1);
   }
}

int main() {
   int num;

   // Prompt user to enter a number
   printf("Enter a non-negative integer: ");
   scanf("%d", &num); //store the number in num

   // Validate the input. Check whether the entered
   // number is non-negative.
   if (num < 0) {
      printf("Factorial is calculated for negative numbers.\n");
   } else {
      // Calculate and print the factorial for input number
      printf("Factorial of %d is %d\n", num, factorial(num));
   }
```

```
    return 0;
}
```

OUTPUT:

```
Enter a non-negative integer: 5
Factorial of 5 is 120
```

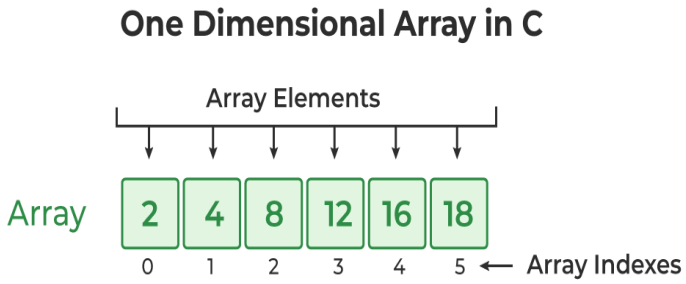
OR

6.a. What is an array? Explain how one dimensional arrays are declared and initialized. Write a C program to find the largest of 'N' elements. [12]

An array is a collection of elements of the same type stored in contiguous memory locations. This organization allows efficient access to elements using their index. Arrays can also be of different types depending upon the direction/dimension they can store the elements. It can be 1D, 2D, 3D, and more. We generally use only one-dimensional, two-dimensional, and three-dimensional arrays.

One-Dimensional Arrays in C

A one-dimensional array can be viewed as a linear sequence of elements. We can only increase or decrease its size in a single direction.



Only a single row exists in the one-dimensional array and every element within the array is accessible by the index. In C, array indexing starts zero-indexing i.e. the first element is at index 0, the second at index 1, and so on up to n-1 for an array of size n.

Syntax of One-Dimensional Array in C

The following code snippets shows the syntax of how to declare an one dimensional array and how to initialize it in C.

1D Array Declaration Syntax

In declaration, we specify then name and the size of the 1d array.

```
elements_type  array_name[array_size];
```

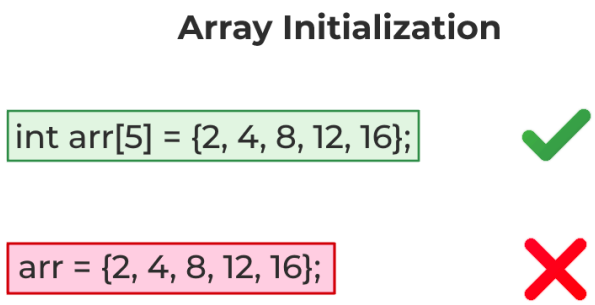
In this step, the compiler reserved the given amount of memory for the array but this step does not define the value of the elements. They may contain some random values. So we initialize the array to give its elements some initial valu

1D Array Initialization Syntax

In declaration, the compiler reserved the given amount of memory for the array but does not define the value of the element. To assign values, we have to initialize an array.

elements_type array_name[array_size] = {value1, value2, ... };
This type of The values will be assigned sequentially, means that first element will contain value1, second value2 and so on.

This initialization only works when performed with declaration.




```
#include <stdio.h>
int main() {
  int n;
  double arr[100];
  printf("Enter the number of elements (1 to 100): ");
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) {
    printf("Enter number%d: ", i + 1);
    scanf("%lf", &arr[i]);
  }
  // storing the largest number to arr[0]
  for (int i = 1; i < n; ++i) {
```

```
  if (arr[0] < arr[i]) {
    arr[0] = arr[i];
  }
 }
 printf("Largest element = %.2lf", arr[0]);
 return 0;
}
```

Output

```
Enter the number of elements (1 to 100): 5
Enter number1: 34.5
Enter number2: 2.4
Enter number3: -35.5
Enter number4: 38.7
Enter number5: 24.5
Largest element = 38.70
```

b. Write a C program to sort the given set of N numbers using Bubble Sort technique. [8]

```
#include <stdio.h>
int main()
{
 int array[100], n, c, d, swap;
 printf("Enter number of elements\n");
 scanf("%d", &n);
 printf("Enter %d integers\n", n);
 for (c = 0; c < n; c++)
  scanf("%d", &array[c]);
 for (c = 0 ; c < n - 1; c++)
 {
  for (d = 0 ; d < n - c - 1; d++)
  {
   if (array[d] > array[d+1]) /* For decreasing order use '<' instead of '>' */
   {
    swap       = array[d];
    array[d]   = array[d+1];
    array[d+1] = swap;
   }
```

```
    }
  }
  printf("Sorted list in ascending order:\n");
  for (c = 0; c < n; c++)
    printf("%d\n", array[c]);
  return 0;
}
```

OUTPUT:

```
Enter number of elements
5
Enter 5 integers
9 3 1 7 0
Sorted list in ascending order:
0
1
3
7
9
```

MODULE 4

7.a. List the applications of arrays. [4]

In the C programming language, arrays are used in a wide range of applications. Few of them are as follows

- Arrays are used to Store List of values
- Arrays are used to Perform Matrix Operations
- Arrays are used to implement Search Algorithms
- Arrays are used to implement Sorting Algorithms
- Arrays are used to implement Data Structures
- Arrays are also used to implement CPU Scheduling Algorithms

b. Write a C program to implement Matrix multiplication and validate the rules of multiplication. [10]

```
#include<stdio.h>

int main()
{
        int a[10][10],b[10][10],c[10][10];
        int m,n,p,q;
        int i,j,k;
```

```
// Input the order of Matrix A - m x n
printf("Enter the order of matrix A :");
scanf("%d%d",&m,&n);

// Input the order of Matrix B - p x q
printf("Enter the order of matrix B:");
scanf("%d%d",&p,&q);

/* For multiplication of two matrices, the
   number of columns in the first matrix
   should be equal to the number of rows
   in the second matrix */
if(n!=p)
{
        printf("Number of columns of Matrix A is not equal to number of rows of matrix B\n");
        printf("Matrix Multiplication not possible....\n");
        return (1);
}

// Input the elements into Matrix A
printf("\nEnter %d elements into matrix A : ", m*n);
for(i=0;i<m;i++)
{
        for(j=0;j<n;j++)
        {
                scanf("%d",&a[i][j]);
        }
}

// Display matrix A in matrix format
printf("\nThe  matrix A is ---\n");
for(i=0;i<m;i++)
{
        for(j=0;j<n;j++)
        {
                printf("%d\t",a[i][j]);
        }
        printf("\n");
}

// Input the elements into Matrix B
printf("\nEnter %d elements into matrix B : ", p*q);
for (i=0;i<p;i++)
{
        for (j=0;j<q;j++)
```

```
            {
                scanf("%d",&b[i][j]);
            }
        }

        // Display Matrix B in matrix format
        printf("\nThe matrix B is ---\n");
        for(i=0;i<p;i++)
        {
            for(j=0;j<q;j++)
            {
                printf("%d\t",b[i][j]);
            }
            printf("\n");
        }

        // Compute (Matrix A) X (Matrix B)
        for(i=0;i<m;i++)
        {
            for(j=0;j<q;j++)
            {
                c[i][j] = 0;
                for(k=0;k<n;k++)
                {
                    c[i][j] = c[i][j] + (a[i][k] * b[k][j]);
                }
            }
        }


        // Display product matrix - Matrix C
        printf("\nThe product matrix is ---\n\n");
        for(i=0;i<m;i++)
        {
            for(j=0;j<q;j++)
            {
                printf("%d\t",c[i][j]);
            }
            printf("\n");
        }

        return 0;
}
```

OUTPUT

```
Enter the order of matrix A :2 2
Enter the order of matrix B:2 2

Enter 4 elements into matrix A : 2 2 2 2

The  matrix A is ---
2       2
2       2

Enter 4 elements into matrix B : 1 1 1 1

The matrix B is ---
1       1
1       1

The product matrix is ---

4       4
4       4
```

c. With syntax and examples, explain the scan set function. [6]

scanf family functions support scanset specifiers **which are represented by %[]**. Inside scanset, we can specify single character or range of characters. While processing scanset, scanf will process only those characters which are part of scanset. We can define scanset by putting characters inside square brackets. Please note that the scansets are case-sensitive.

We can also use scanset by providing comma in between the character you want to add.

Syntax:

```
scanf("%[characters/range]",name_scanf);
```

Example: scanf(%s[A-Z,_,a,b,c]s,str);

```
/* A simple scanset example */
#include <stdio.h>

int main(void)
{
   char str[128];
```

```
    printf("Enter a string: ");
    scanf("%[A-Z]s", str);

    printf("You entered: %s\n", str);

    return 0;
}
```

Output:

```
Enter a string: Skill Vertex
You entered: Skill Vertex
```

OR

8.a. Explain the different methods of reading and writing strings using formatted and unformatted functions. Write an example for each. [12]

Formatted and Unformatted I/O Statements/functions
Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user.
Formatted I/O functions
1. printf()
2. scanf()
3. sprintf()
4. sscanf()

1. printf():
printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file).
Syntax :
To display any variable value.
printf("Format Specifier", var1, var2, …., varn);
Example:
printf("Enter the text which you want to display");
Output:
Enter the text which you want to display

Or

```
#include&lt;stdio.h&gt;
void main()
{
int a;
```

a = 20; // Assigning a value in a variable
printf("%d", a); // Printing the value of a variable
}
Output:
20

2. scanf():
scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file).

Syntax:
scanf("Format Specifier", &var1, &var2, …., &varn);

Example:
#include<stdio.h>
int main()
{
int num1;
printf("Enter a integer number: ");
scanf("%d", &num1);
printf("You have entered %d", num1);
return 0;
}
Output:
Enter a integer number: 56
You have entered 56
3. sprintf():
sprintf stands for "string print". This function is similar to printf() function but this function prints the string into a character array instead of printing it on the console screen.
Syntax:
sprintf(array_name, "format specifier", variable_name);
Example:
#include <stdio.h>
int main()
{
char str[50];
int a = 2, b = 8; // The string "2 and 8 are even number"
sprintf(str, "%d and %d are even number", a, b); // is now stored into str
printf("%s", str); // Displays the string
return 0;
}
Output
2 and 8 are even number

4. sscanf():
sscanf stands for “string scanf”. This function is similar to scanf() function but this function reads data from the string or character array instead of the console screen.
Syntax:
sscanf(array_name, “format specifier”, &variable_name);
Example:

```
#include &lt;stdio.h&gt;
int main()
{
char str[50];
int a = 2, b = 8, c, d; // The string &quot;a = 2 and b = 8&quot;
sprintf(str, &quot;a = %d and b = %d&quot;,a, b); // is now stored into str , character array
sscanf(str, &quot;a = %d and b = %d&quot;,&amp;c, &amp;d); // The value of a and b is now in c and d
printf(&quot;c = %d and d = %d&quot;, c, d); // Displays the value of c and d
return 0;
}
```

Output:
c = 2 and d = 8

Unformatted Input/Output Statements/functions
Unformatted I/O functions are used only for character data type or character array/string and cannot be used for any other datatype. These functions are used to read single input from the user at the console and it allows to display the value at the console.
These functions are called unformatted I/O functions because we cannot use format specifiers in these functions and hence, cannot format these functions according to our needs.
Unformatted Input/Output Statements
1. getch()
2. getche()
3. getchar()
4. putchar()
5. gets()
6. puts()
7. putch()

1. getch():
getch() function reads a single character from the keyboard by the user but doesn’t display that character on the console screen and immediately returned without pressing enter key. This function is declared in conio.h(header file). getch() is also used for hold the screen.
Syntax:
getch();
or
variable-name = getch();
Example:

```
#include &lt;conio.h&gt;
#include &lt;stdio.h&gt;
```

int main()
{
printf("Enter any character: ");
getch();// Reads a character but & not displays
return 0;
}
Output:
Enter any character:
2. getche():
getche() function reads a single character from the keyboard by the user and displays it on the console screen and immediately returns without pressing the enter key. This function is declared in conio.h(header file).
Syntax:
getche();
or
variable_name = getche();
Example:
#include <conio.h>
#include <stdio.h>
int main()
{
printf("Enter any character: "); // Reads a character and displays immediately
getche();
return 0;
}
Output:
Enter any character: g

3. getchar():
The getchar() function is used to read only a first single character from the keyboard whether multiple characters is typed by the user and this function reads one character at one time until and unless the enter key is pressed. This function is declared in stdio.h(header file)
Syntax:
Variable-name = getchar();
Example:
#include <conio.h>
#include <stdio.h>
int main()
{
char ch; // Declaring a char type variable
printf("Enter the character: ");
ch = getchar(); // Taking a character from keyboard
printf("%c", ch); // Displays the value of ch
return 0;
}

Output:
Enter the character: a
a
4. putchar():
The putchar() function is used to display a single character at a time by passing that character directly to it or by passing a variable that has already stored a character. This function is declared in stdio.h(header file)
Syntax:
putchar(variable_name);
Example:
#include <conio.h>
#include <stdio.h>
int main()
{
char ch;
printf("Enter any character: ");
ch = getchar(); // Reads a character
putchar(ch); // Displays that character
return 0;
}
Output:
Enter any character: Z
Z

5. gets():
gets() function reads a group of characters or strings from the keyboard by the user and these characters get stored in a character array. This function allows us to write space-separated texts or strings. This function is declared in stdio.h(header file).
Syntax:
char str[length of string in number]; //Declare a char type variable of any length
gets(str);
Example:
#include <conio.h>
#include <stdio.h>
int main()
{
char name[50]; // Declaring a char type array of length 50 characters
printf("Please enter some texts: ");
gets(name); // Reading a line of character or a string
printf("You have entered: %s",name); // Displaying this line of character or a string
return 0;
}
Output:
Please enter some texts: Welcome to CCP/POP
You have entered: Welcome to CCP/POP

6. puts():
puts() function is used to display a group of characters or strings which is already stored in a character array. This function is declared in stdio.h(header file).
Syntax:
puts(identifier_name);
Example:
#include <stdio.h>
int main()
{
char name[50];
printf("Enter your text: ");
gets(name); // Reads string from user
printf("Your text is: ");
puts(name); // Displays string
return 0;
}
Output:
Enter your text: GeeksforGeeks
Your text is: GeeksforGeeks

7. putch():
putch() function is used to display a single character which is given by the user and that character prints at the current cursor location. This function is declared in conio.h(header file)
Syntax:
putch(variable_name);
Example:
#include <conio.h>
#include <stdio.h>
int main()
{
char ch;
printf("Enter any character:\n ");
ch = getch(); // Reads a character from the keyboard
printf("\nEntered character is: ");
putch(ch); // Displays that character on the console
return 0;
}
Output:
Enter any character:
Entered character is: d

b. Write a C program to pass a two dimensional array to the function and display in matrix format. [8]

#include <stdio.h>

```
#define ROWS 3
#define COLS 4

// Function to display the matrix
void displayMatrix(int matrix[ROWS][COLS], int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    // Initialize a 2D array
    int matrix[ROWS][COLS] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };

    // Display the matrix
    printf("The matrix is:\n");
    displayMatrix(matrix, ROWS, COLS);

    return 0;
}
```

OUTPUT:

```
The matrix is:
1       2       3       4
5       6       7       8
9       10      11      12
```

MODULE 5

9.a. Explain the following string manipulation functions: [8]
(i) strlen() (ii) strcpy() (iii) strcmp() (iv) strcat()

String manipulation functions in C are part of the <string.h> library and are used to perform various operations on strings (character arrays). Let's go through the functions you've mentioned:

(i) strlen()

Function:

- The strlen() function is used to calculate the length of a string (i.e., the number of characters in the string before the null terminator '\0').

Syntax:

size_t strlen(const char *str);

Parameters:

- str: A pointer to the first character of the string whose length is to be determined.

Returns:

- The function returns the length of the string as an integer (size_t), excluding the null terminator.

Example:

```
#include <stdio.h>

#include <string.h>

int main() {

   char str[] = "Hello, World!";

   printf("Length of the string is: %lu\n", strlen(str));

   return 0;

}
```

Output:

Length of the string is: 13

(ii) strcpy()

Function:

- The strcpy() function is used to copy a string from the source (src) to the destination (dest).

Syntax:

char *strcpy(char *dest, const char *src);

Parameters:

- dest: A pointer to the destination array where the content will be copied.
- src: A pointer to the source string to be copied.

Returns:

- The function returns a pointer to the destination string (dest).

Example:

```
#include <stdio.h>

#include <string.h>

int main() {

   char src[] = "Hello, World!";

   char dest[20];

   strcpy(dest, src);

   printf("Copied string: %s\n", dest);

   return 0;

}
```

Output:

Copied string: Hello, World!

(iii) strcmp()

Function:

- The strcmp() function is used to compare two strings (str1 and str2).

Syntax:

int strcmp(const char *str1, const char *str2);

Parameters:

- str1: A pointer to the first string to be compared.
- str2: A pointer to the second string to be compared.

Returns:

- The function returns an integer that can be:
  - **0**: If str1 and str2 are equal.
  - **< 0**: If str1 is less than str2.
  - **> 0**: If str1 is greater than str2.

Example:

```
#include <stdio.h>

#include <string.h>

int main() {

   char str1[] = "Hello";

   char str2[] = "World";

   int result = strcmp(str1, str2);

   if (result == 0) {

      printf("The strings are equal.\n");

   } else if (result < 0) {

      printf("str1 is less than str2.\n");

   } else {
```

```
    printf("str1 is greater than str2.\n");

  }

  return 0;

}
```

Output:

str1 is less than str2.

(iv) strcat()

Function:

- The strcat() function is used to concatenate (append) the source string (src) to the end of the destination string (dest).

Syntax:

char *strcat(char *dest, const char *src);

Parameters:

- dest: A pointer to the destination array, which should contain a string and have enough space to hold the result.
- src: A pointer to the source string to be appended.

Returns:

- The function returns a pointer to the resulting string (dest).

Example:

```
#include <stdio.h>

#include <string.h>

int main() {

  char dest[20] = "Hello";
```

```
    char src[] = ", World!";

    strcat(dest, src);

    printf("Concatenated string: %s\n", dest);

    return 0;

}
```

Output:

Concatenated string: Hello, World!

b. Define pointer. Explain the declaration and initialization of a pointer variable with an example. [4]

Definition:
A pointer is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers.

Syntax of C Pointers
The syntax of pointers is similar to the variable declaration in C, but we use the **(*) dereferencing operator** in the pointer declaration.
datatype * ptr;

1. Pointer Declaration

In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the **(*) dereference operator** before its name.

Example

int *ptr;

The pointer declared here will point to some random memory address as it is not initialized. Such pointers are called wild pointers.

2. Pointer Initialization

Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the **(&: ampersand) addressof operator** to get the memory address of a variable and then store it in the pointer variable.

Example

```
int var = 10;
int * ptr;
ptr = &var;
```

Write a C program to compute the sum mean and standard deviation of all elements stored in an array of N real numbers using pointers.

c. [8]

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

int main()
{
        int i, n;
        float a[10],sum,mean,var,sd;
        float *p;       // p is a pointer to float data

        // Accept number of elements - n
        printf("Enter Number of elements:");
        scanf("%d", &n);

        // Accept n real numbers
        printf("Enter %d numbers :",n);
        p = a;              // pointer p points to first element of a
        for (i=0;i<n;i++)
        {
                scanf("%f", p);
                p++;    // pointer p points to the next element of a
        }

        // Compute Sum of array elements
        sum = mean = var = sd = 0.0;
        p = a;              // pointer p points to the first element of a
        for(i=0;i<n;i++)
        {
                sum = sum + (*p);
                p++;
        }

        // Compute mean
```

```
        mean = sum / (float) n;

        // Compute Variance
        p = a;
        for(i=0;i<n;i++)
        {
                var = var + pow( (*p - mean),2);
                p++;
        }
        var = var / (float) n;

        // Compute Standard Deviation
        sd = sqrt(var);

        // print sum, mean and standard deviation
        printf("Sum = %f\n", sum);
        printf("Mean = %f\n", mean);
        printf("Standard Deviation = %f\n", sd);

        return 0;
}
```

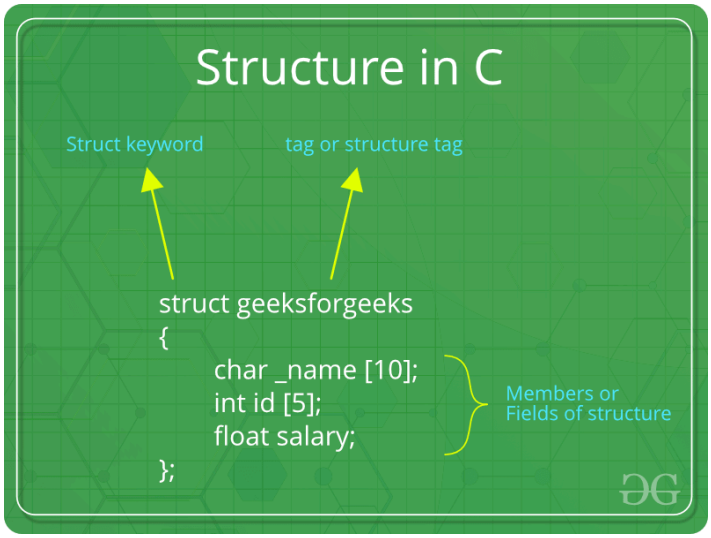
OUPUT

```
Enter Number of elements:5
Enter 5 numbers :1
2
3
4
5
Sum = 15.000000
Mean = 3.000000
Standard Deviation = 1.414214
```

OR

10.a Define structure. Explain the declaration of structure with an example. [8]

The structure in C is a user-defined data type that can be used to group items of possibly different types into a single type. The **struct keyword** is used to define the structure in the C programming language. The items in the structure are called its **member** and they can be of any valid data type. Additionally, the values of a structure are stored in contiguous memory locations.



C Structure Declaration

We have to declare structure in C before using it in our program. In structure declaration, we specify its member variables along with their datatype. We can use the struct keyword to declare the structure in C using the following syntax:

Syntax

```
struct structure_name {
  data_type member_name1;
  data_type member_name1;
  ....
  ....
};
```

The above syntax is also called a structure template or structure prototype and no memory is allocated to the structure in the declaration.

Example:

```
#include <stdio.h>

// Declaration of the structure
struct Person {
    char name[50];  // Name of the person
    int age;        // Age of the person
    float height;   // Height of the person in meters
};

int main() {
    // Declare a variable of type 'struct Person'
    struct Person person1;
```

```
    // Assign values to the members of the structure
    printf("Enter the name of the person: ");
    scanf("%s", person1.name);
    printf("Enter the age of the person: ");
    scanf("%d", &person1.age);
    printf("Enter the height of the person in meters: ");
    scanf("%f", &person1.height);

    // Access and print the values of the structure members
    printf("\nPerson Details:\n");
    printf("Name: %s\n", person1.name);
    printf("Age: %d\n", person1.age);
    printf("Height: %.2f meters\n", person1.height);

    return 0;
}
```

Output:

```
Enter the name of the person: John
Enter the age of the person: 25
Enter the height of the person in meters: 1.75

Person Details:
Name: John
Age: 25
Height: 1.75 meters
```

b. [12]

Write a C program to implement structure to read, write and compute average marks and the students scoring above and below the average marks for a class of N students.

```
#include<stdio.h>

 struct student
        {
                int id;
                char name[20];
                float sub[6];
                float avg;
        };

 int main()
 {
        struct student s[20];
        float sum=0;
```

```
int i,j,n;

// Accept the number of records/students
printf("Enter the number of records :");
scanf("%d",&n);

// Accept data for all the fields/members of each record
printf("Enter %d student details...\n",n);

for(i=0;i<n;i++)
{
        printf("\n\nEnter student ID, name :");          // Student ID
        scanf("%d%s",&s[i].id, s[i].name);

        printf("Enter 6 subject marks :");

        for (j=0;j<6;j++)
                {
                        scanf("%f", &s[i].sub[j]);
                }
}

// Compute the average of each student

for(i=0;i<n;i++)
{
        sum=0;
        for (j=0;j<6;j++)
                {
                        sum = sum + s[i].sub[j];
                }
        s[i].avg = sum / 6;
}

// Display student ID, name and average of all students
// who have scored above average marks
printf("Students scoring above the average marks....\n");
printf("\n\nID\tName\tAverage\n\n");

for(i=0;i<n;i++)
{
        if(s[i].avg>=35.0)
        printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
}
```

```
        // Display student ID, name and average of all students
        // who have scored below average marks

        printf("\n\nStudents scoring below the average marks....\n");
        printf("\n\nID\tName\tAverage\n\n");

        for(i=0;i<n;i++)
        {
                if(s[i].avg<35.0)
                printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
        }

        return 0;
 }
```

OUTPUT:

```
Enter the number of records :3
Enter 3 student details...

Enter student ID, name :1 hema
Enter 6 subject marks :90 89 78 99 100 98

Enter student ID, name :2 Sanvi
Enter 6 subject marks :66 65 45 23 42 58

Enter student ID, name :3 Pavi
Enter 6 subject marks :23 25 51 62 24 23
Students scoring above the average marks....

ID      Name  Average

1       hema  92.333336
2       Sanvi 49.833332

Students scoring below the average marks....

ID      Name  Average

3       Pavi    34.666668
```